

IDC DOCUMENTATION

**Continuous Data
Subsystem
CD-1.1
Software
User Manual**



Notice

This document was published January 2001 by the Monitoring Systems Operation of Science Applications International Corporation (SAIC) as part of the International Data Centre (IDC) Documentation. Every effort was made to ensure that the information in this document was accurate at the time of publication. However, information is subject to change.

Contributors

Raymond L. Cordova, Science Applications International Corporation

Trademarks

ORACLE is a registered trademark of Oracle Corporation.

SAIC is a trademark of Science Applications International Corporation.

Solaris is a registered trademark of Sun Microsystems.

SQL*Plus is a registered trademark of Oracle Corporation.

Sun is a registered trademark of Sun Microsystems.

UNIX is a registered trademark of UNIX System Labs, Inc.

Ordering Information

The ordering number for this document is SAIC-01/3006.

This document is cited within other IDC documents as [IDC6.5.18].

Continuous Data Subsystem CD-1.1 Software User Manual

CONTENTS

<u>About this Document</u>	i
■ <u>PURPOSE</u>	ii
■ <u>SCOPE</u>	ii
■ <u>AUDIENCE</u>	iii
■ <u>RELATED INFORMATION</u>	iii
■ <u>USING THIS DOCUMENT</u>	iii
<u>Conventions</u>	v
<u>Chapter 1: Introduction</u>	1
■ <u>SOFTWARE OVERVIEW</u>	2
■ <u>STATUS OF DEVELOPMENT</u>	5
■ <u>FUNCTIONALITY</u>	5
<u>Performance Characteristics</u>	5
■ <u>INVENTORY</u>	8
■ <u>ENVIRONMENT AND STATES OF OPERATION</u>	12
<u>Software Environment</u>	12
<u>Normal Operational State</u>	13
<u>Chapter 2: Operational Procedures</u>	15
■ <u>SOFTWARE STARTUP</u>	16
<u>Automated Invocations</u>	16
<u>Nonautomated Invocations</u>	20
<u>Manual Methods</u>	21
■ <u>SOFTWARE SHUTDOWN</u>	25
<u>Shutdown for Automated Invocations</u>	25
<u>Shutdown for Nonautomated Invocations</u>	27
■ <u>MAINTENANCE</u>	29

■ SECURITY	30
Chapter 3: Troubleshooting	31
■ MONITORING	32
Inbound Data and Connections	32
Processes Controlled by Data Center Manager	37
■ INTERPRETING ERROR MESSAGES	43
Connection Manager	43
Connection Manager Server	63
Connection Originator	74
Data Parser	77
Data Center Manager	82
Frame Exchange	89
Exchange Controller	94
Protocol Checker	102
■ SOLVING COMMON PROBLEMS	103
Error Recovery	103
■ REPORTING PROBLEMS	104
Chapter 4: Installation Procedures	105
■ PREPARATION	106
Obtaining Released Software	106
Hardware Mapping	106
UNIX System	106
Firewall	109
■ EXECUTABLE FILES	110
■ DATA STORES	111
Log Files	111
Creating Stores—Logging, Frame Store, and Disk Loops	112
■ CONFIGURATION DATA FILES	113
Application Configuration Overview	113
Creating Run-time Environments	115
Subsystem-level Configuration	116

Connection Manager/Connection Manager Server	117
Connection Originator	119
Frame Exchange/Exchange Controller	120
Data Parser	123
Frame Store	129
Data Center Manager	133
Authentication	134
■ DATABASE	135
Accounts	135
Tables	135
■ SHORT COURSE FOR INSTALLING SOFTWARE	136
Shutdown of Running CDS CD-1.1	137
Install	137
Restart CDS CD-1.1	140
■ ADDING DATA PROVIDERS	140
Frame Store Changes	141
Connection Manager Changes	143
Consumer Frame Exchange/Exchange Controller Setup	143
Forwarding Frame Exchange/Exchange Controller Setup	144
Authentication Updates	144
Database Definitions	145
Environment Modification	151
■ SHORT COURSE FOR ADDING DATA PROVIDERS	152
Frame Store Changes	153
Connection Manager Changes	153
Forwarding Frame Exchange/Exchange Controller Setup	153
Data Parser Changes	154
Authentication Updates	154
Database Definitions	154
Environment Modifications	154
■ ADDING FORWARDING DESTINATION	155
Subsystem-level Changes	155
Frame Store Changes	156

Connection Originator Setup	157
Frame Exchange/Exchange Controller Setup	158
Data Center Manager Setup	160
Authentication Updates	162
Environment Modification	163
References	165
Appendix: Tools	A1
■ ENVIRONMENT CREATION	A2
■ FRAME SET MAKER	A7
■ MAKE LOOP (MKLOOP)	A8
Glossary	G1
Index	I1

Continuous Data Subsystem CD-1.1 Software User Manual

FIGURES

<u>FIGURE 1.</u>	<u>IDC SOFTWARE CONFIGURATION HIERARCHY</u>	3
<u>FIGURE 2.</u>	<u>RELATIONSHIP OF CONTINUOUS DATA SUBSYSTEM TO DATA SERVICES</u>	4
<u>FIGURE 3.</u>	<u>CDS CD-1.1 ARCHITECTURAL DATA FLOW</u>	11
<u>FIGURE 4.</u>	<u>CDS CD-1.1 PROCESS INVOCATION</u>	17
<u>FIGURE 5.</u>	<u>CONNECTION STATE TRANSITIONS</u>	19
<u>FIGURE 6.</u>	<u>SAMPLE PAR FILE HIERARCHY</u>	114

Continuous Data Subsystem CD-1.1 Software User Manual

TABLES

<u>TABLE I:</u>	<u>DATA FLOW SYMBOLS</u>	v
<u>TABLE II:</u>	<u>TYPGRAPHICAL CONVENTIONS</u>	vi
<u>TABLE 1:</u>	<u>DATABASE TABLES AND CDS CD-1.1 APPLICATIONS</u>	136

About this Document

This chapter describes the organization and content of the document and includes the following topics:

- [Purpose](#)
- [Scope](#)
- [Audience](#)
- [Related Information](#)
- [Using this Document](#)

About this Document

PURPOSE

This document describes how to use the Continuous Data Subsystem for CD-1.1 (*CDS CD-1.1*) software at a data center. The data center may be the International Data Centre (IDC), the Prototype International Data Centre (PIDC), or some other data center that receives and forwards continuous data following the Continuous Data 1.1 (CD-1.1) formats and protocols. Continuous Data Subsystem for CD-1.1 is a computer software component (CSC) of the Data Services Computer Software Configuration Item (CSCI). The Continuous Data Subsystem for CD-1.1 software is identified as follows:

Title: Continuous Data Subsystem for CD-1.1

Abbreviation: *CDS CD-1.1*

SCOPE

This manual includes instructions for setting up the software, using its features, and basic troubleshooting. This document does not describe the software's design or requirements. These topics are described in sources cited in "[Related Information](#)."

Two basic protocols are in use: CD-1.0, formerly known as the Alpha protocol, and CD-1.1. The software described in this document is solely devoted to handling data in the CD-1.1 protocol.

AUDIENCE

This document is intended for the first-time or occasional user of the software. However, more experienced users may find certain sections useful as a reference.

RELATED INFORMATION

The following documents complement this document:

- *Formats and Protocols for Continuous Data CD-1.1* [\[IDC3.4.3\]](#)
- *Continuous Data Subsystem CD-1.1* [\[IDC7.4.1\]](#)

See [“References” on page 165](#) for a list of documents that supplement this document. The following UNIX manual (man) pages apply to the *CDS CD-1.1* software:

- *ConnMgr*
- *ConnMgr_server*
- *ConnOrig*
- *DLParse*
- *ExCtrlr*
- *ForeMan*
- *FrameEx*
- *ProtoCheck*
- *libas*
- *libcdo*
- *libfio*
- *libfs*

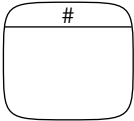

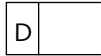
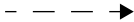

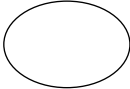
USING THIS DOCUMENT

This document is part of the overall documentation architecture for the IDC. The document is part of the Technical Instructions category, which provides guidance for installing, operating, and maintaining the IDC systems. This document is organized as follows:

Conventions

This document uses a variety of conventions, which are described in the following tables. [Table I](#) shows the conventions for data flow diagrams. [Table II](#) lists typographical conventions.

TABLE I: DATA FLOW SYMBOLS

Description	Symbol
process	
external source or sink of data	
disk store	
control flow	
data flow	
state	

▼ About this Document

TABLE II: TYPOGRAPHICAL CONVENTIONS

Element	Font	Example
database table	bold	wfproto
database attributes	<i>italics</i>	<i>dlid</i>
processes, software units, and libraries		<i>ConnMgr</i>
user-defined arguments and variables used in parameter (par) files or program com- mand lines		<code>run_idc_dcmgr <hostname></code>
titles of documents		<i>Continuous Data Subsystem CD-1.1</i>
computer code and output filenames, directories, and websites	<code>courier</code>	<code>[info]:Parameter inetd - 1</code> <code>DLlog.log</code>
text that should be typed exactly as shown		<code>ps -fu <cds-user-name></code>

Chapter 1: Introduction

This chapter provides a general description of the software and includes the following topics:

- [Software Overview](#)
- [Status of Development](#)
- [Functionality](#)
- [Inventory](#)
- [Environment and States of Operation](#)

Chapter 1: Introduction

SOFTWARE OVERVIEW

The Continuous Data Subsystem for CD-1.1 (*CDS CD-1.1*) contains software that implements the CD-1.1 protocol [\[IDC3.4.3\]](#). The CD-1.1 protocol is a set of formats, protocols, and policies that define how to send continuous, time-series data from one computer to another. Physical transmission of data may occur over land-lines, through radio connections, and via satellite links. Transmissions over these physical links will be in the form of Data Frames constructed and exchanged according to the protocol document. The CD-1.1 protocol exists at the application layer in the Internet Protocol (IP) communications layered model. Elements of the *CDS CD-1.1* will comply, via hardware and software solutions, with the formats and protocol document and system requirements to reliably deliver data in support of the monitoring system mission. In the context of the nuclear treaty monitoring, these data are acquired for seismic, hydroacoustic, infrasonic, or other environmental sensors. The *CDS CD-1.1* application software will execute at the IDC. The *CDS CD-1.1* software is responsible for acquiring data from providers, forwarding data to subscribing National Data Centers (NDCs), and making received data available for signal processing and analysis software at the IDC.

The design of the *CDS CD-1.1* software is such that its execution is cooperative and nonintrusive with the operation of CDS software components that support the CD-1.0 protocol. Components that support the CD-1.0 protocol are not addressed by this document. Continuous Data Subsystem CD-1.0 is a separate suite of software components that implement the CD-1.0 formats and protocols [IDC3.4.2].

[Figure 1](#) shows the logical organization of the IDC software. The Continuous Data Subsystem for CD-1.1 is one component of the Data Services CSCI. [Figure 2](#) shows a processing flow model of the IDC system and the relationship of the Continuous Data Subsystem for CD-1.1 to other components of the system.

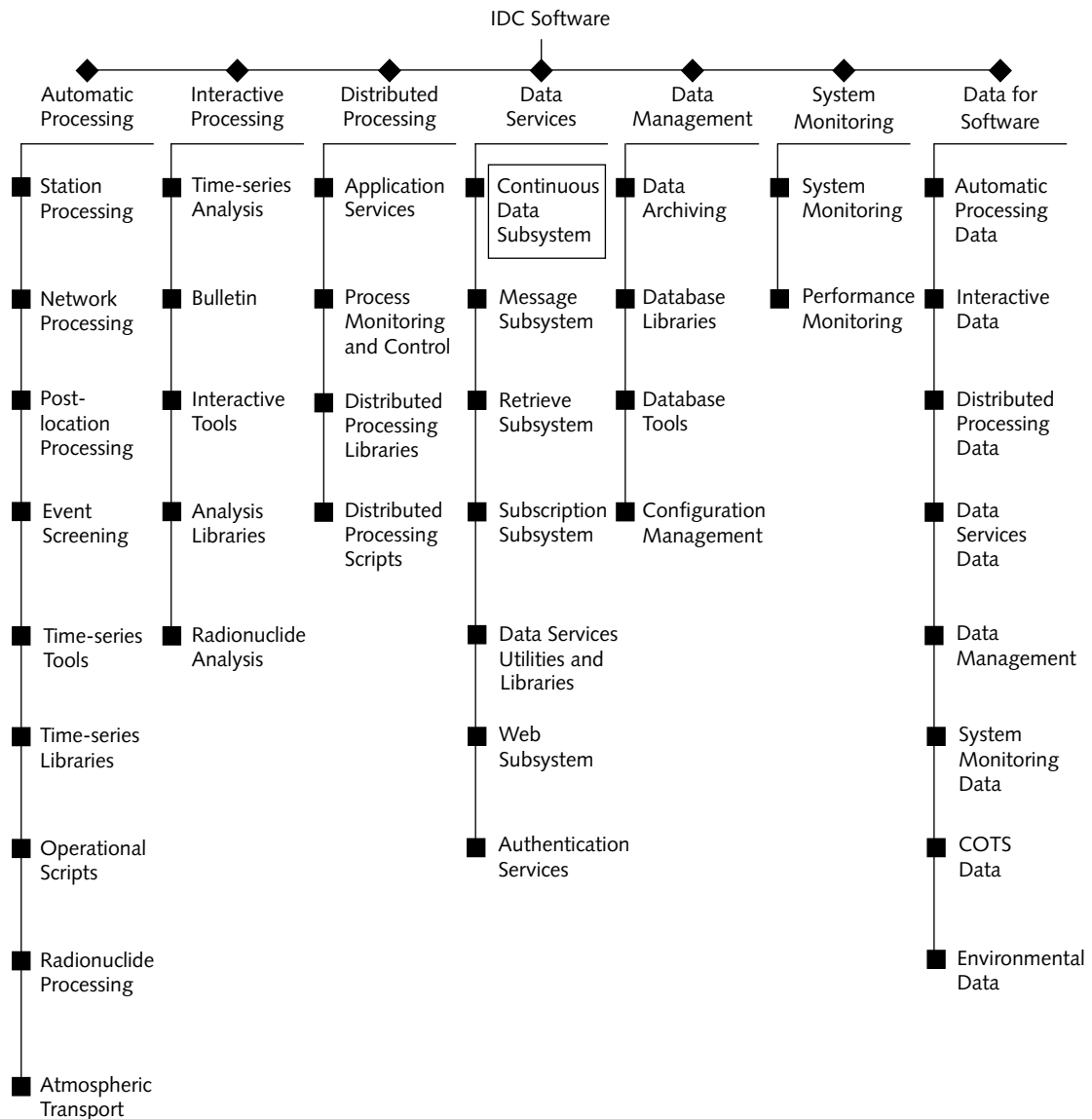


FIGURE 1. IDC SOFTWARE CONFIGURATION HIERARCHY

▼ Introduction

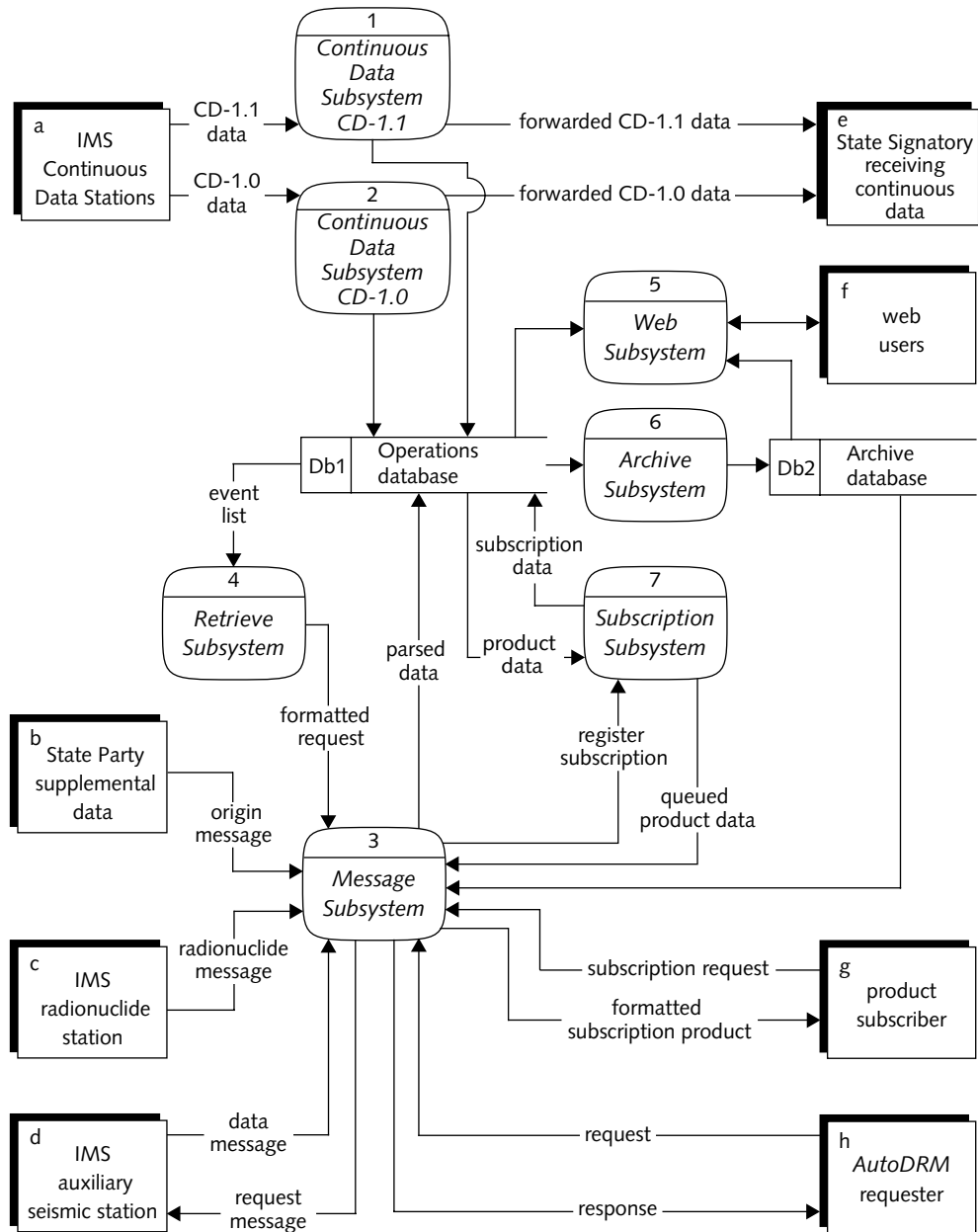


FIGURE 2. RELATIONSHIP OF CONTINUOUS DATA SUBSYSTEM TO DATA SERVICES

STATUS OF DEVELOPMENT

The *CDS CD-1.1* was designed and developed during 1999 and 2000 to implement the CD-1.1 formats, protocols, and policies [\[IDC3.4.3\]](#). Design of the *CDS CD-1.1* software suite is complete and is presented in the *Continuous Data Subsystem CD-1.1 Software Design Document* [\[IDC7.4.1\]](#). The components of the Continuous Data Subsystem for CD-1.1 take advantage of existing software libraries, methods, and configurations currently in use by the operational software system. Reuse of the software aids in the installation, maintainability and operability of the *CDS CD-1.1*.

The software was installed at the PIDC in September 2000. At that time, no stations transmitted data following the Continuous Data Subsystem for CD-1.1 protocol. Therefore, *CDS CD-1.1* was tested with a synthetic Data Frame generator and with hardware of the Malin, Ukraine array operating in an integration facility.

FUNCTIONALITY

The *CDS CD-1.1* provides the system with the ability to receive time-series signal data from providing sources via a continuous data protocol. Receiving data entails receipt of protocol frames, data authentication, and storing, parsing, and forwarding received data. Parsing data refers to the processing that makes time-series signal data available for use by other processing components of the system. Forwarding data results in sending data packets/frames received from providers to other data centers, that is, consumers of data.

Performance Characteristics

The *CDS CD-1.1* software is designed to support the flow of continuous data into the IDC and out of the IDC in the form of protocol frames. The performance of the software and the load it will place on the hosting computers is governed by the number of sites providing CD-1.1 data and the volume from each one of those sites. However, some evaluations and expectations may be stated.

▼ Introduction

Inbound data to the *CDS CD-1.1* software will be CD-1.1 protocol frames provided via Transmission Control Protocol/IP (TCP/IP) socket connections. Processing of the *CDS CD-1.1* software does not attempt to throttle input over an established connection. That is, data will be accepted at the rate of the processing available on the host computer and the available bandwidth of the network connection. The processing load for supporting a single provider producing 20 KB frames at 0.1 Hz will not tax system performance. When many providers are being supported and in transitory conditions, processing loads could be significant. Of particular interest are the effects of authentication and provider startup.

Authentication processing is a computationally intensive activity that is associated with each signed input stream. The processing load increases linearly as the number of input streams increases. Experience has shown that receiving and parsing protocol frames is an inexpensive activity as compared with authentication. No tricks are available to diminish the cost of Channel Subframe authentication short of disabling authentication.

When a connection is established the CD-1.1 protocol requires that the provider attempt to send all frames that have not been previously sent and acknowledged. If the connection has been broken for a long period of time, a great number of frames may be sent as part of the startup processing. This data delivery behavior has been designed in to the software and will manifest itself in heavy processor utilization. This will be a transitory condition that will abate after the initial back-log of frames has been delivered.

Two outbound flows of data from the *CDS CD-1.1* are used. The first outbound flow is to forwarding destinations. Like the inbound flow, this will be CD-1.1 protocol frames sent via TCP/IP socket connections and consist primarily of protocol Data Frames. The second outbound flow is to the Database Management System (DBMS) and file system for waveform description (**wfdisc**) records and waveform data, respectively.

Frames are forwarded by instantiations of forwarding streams, where one or more separate forwarding streams are required for each forwarding site/destination. The decision on the number of forwarding streams to define per forwarding site is driven by the amount of data to be forwarded and will influence the host processor

load. As an example, suppose 100 sites provide data, and NDC-A is to receive forwarded data from all 100. If an otherwise lightly loaded machine is available, it may be reasonable to have all 100 sites forwarded by a single instantiation of forwarding software. Alternatively, two computers and two instantiations of forwarding software could be used, each forwarding data from 50 sites.

The second outbound flow from the *CDS CD-1.1* contains data that have been converted into a form suitable for other processing applications. Like the outbound flow of forwarded frames, this flow may also be configured to be generated by a single or multiple instantiation of the processing software. The conversion and writing of data to the DBMS and the file system is efficient. A given parsing/conversion instantiation should have no problem keeping up with a minimum of 20 stations. However, the software that converts data is also responsible for authentication of digital signatures on Channel Subframes. Signature authentication is a computationally intensive activity involving many mathematical operations per signature. Whereas performance of inbound and forwarding outbound processing is input/output (I/O) bound, performance of conversion software is Central Processing Unit (CPU) bound. Based on the processing speed of the host computer, the 20 stations per parsing instantiation mentioned above may be more than can be accommodated on a host computer when authentication is applied.

The components of the *CDS CD-1.1* are highly configurable allowing the user to customize the processing environment. The ability to customize carries with it a responsibility to understand the implications of configuration decisions. Therefore, it is reasonable to expect that the performance of the *CDS CD-1.1* should keep up with the near real-time flow of data from an International Monitoring System (IMS) network of 150 sites, all providing data according to the CD-1.1 protocol. The design of the subsystem supports this. However, configuration choices will influence the ability to do so. The following configuration issues are of particular interest with respect to performance:

- number of inbound sources (sites) hosted on a computer
- number of forwarded sites to a given destination being hosted on a computer
- number of sources being parsed/converted on a given computer

▼ Introduction

- location of Frame Stores in the Network File System (NFS) relative to the host computers processing data stored in the Frame Store
- location of Disk Loops in the NFS relative to the host computer producing data for the Disk Loops

The *CDS CD-1.1* software is designed to provide reliable data receipt and data forwarding. To accomplish this, the software contains components that monitor the execution of processes and re-fork tasks that exit. Additionally, the Frame Store provides a durable store to ensure that data are not lost during processing backlogs or communication outages. On both inbound and outbound data flows, a user can expect to have a greater than 99 percent data delivery of protocol frames.

INVENTORY

The following list presents the executables of the *CDS CD-1.1* and their functions. [Figure 3](#) shows the architectural data flow amongst these executables.

- *Connection Manager*
The *Connection Manager* (*ConnMgr*) is responsible for responding to requests to provide *CDS CD-1.1* data. Connection requests may be from IMS stations or from another data center that is providing data (in other words, from any source of continuous data). Only one *Connection Manager* is needed for an entire data center. The *Connection Manager* is capable of distributing inbound connections on the Local Area Network (LAN) so that more than one computer may be used for receiving the CD-1.1 protocol data. This is accomplished with the cooperation of the *Connection Manager Server* process, which is instantiated on each host servicing a connection.
- *Connection Manager Server*
The *Connection Manager Server* (*ConnMgr_server*) is responsible for servicing requests from the *Connection Manager* to support/establish a connection to an inbound request to provide the *CDS CD-1.1* data. A *Connection Manager Server* instance is needed on each computer that will host an inbound connection, but is not used for outbound/forwarding

connections. The *Connection Manager Server* requires LAN connectivity to the computer hosting the *Connection Manager* as well as Wide Area Network (WAN) connectivity for data provider connections.

- *Connection Originator*

The *Connection Originator* (*ConnOrig*) is responsible for establishing a connection to a data consumer, that is, the destination of data forwarding. The *Connection Originator* requires connectivity to the WAN on which forwarding connections take place.

- *Data Parser*

The *Data Parser* (*DLParse*) is responsible for parsing newly received Data Frames, writing time-series data to disk-loop files (in CSS 3.0 format), and updating waveform description (**wfdisc**) records in the database.

- *Exchange Controller*

The *Exchange Controller* (*ExCtrl*) is responsible for communication links between CD-1.1 data providers and consumers. *Exchange Controller* provides the policy control for protocol frame delivery and receipt and works in conjunction with the *Frame Exchange* (*FrameEx*) to deliver and receive frames.

- *ForeMan*

The *Data Center Manager* (*ForeMan*) is responsible for starting, monitoring, and restarting the CDS CD-1.1 processes at the data center. Processes that are started are *Connection Originator* for forwarding data streams and *Data Parser* for incoming data streams. After *Connection Originator* has successfully established a connection, it will become the *Exchange Controller* process as far as *ForeMan* is concerned. *ForeMan* will monitor *Exchange Controller* and *Data Parser* and provide restart capability if needed. If *ExCtrl* requires restarting, *ForeMan* will restart *Connection Originator* as before. One *ForeMan* process for each computer is used to forward data or host a *Data Parser* process.

▼ Introduction

■ *Frame Exchange*

The *Frame Exchange* (*FrameEx*) is responsible for providing low-level processing for reliable delivery and receipt of protocol frames. *FrameEx* works in conjunction with the *Exchange Controller*, reacting to send requests and providing notification of frame receipt. *FrameEx* requires network connectivity to the WAN for interacting with data providers and data receivers.

■ *Protocol Checker*

The *Protocol Checker* (*ProtoCheck*) is a utility program that examines frames in a frame set for compliance with protocol frame definition. *Protocol Checker* is not part of the operational Continuous Data Subsystem.

■ *cds_idc_env* is a C-shell script for defining environment variables needed for the CDS CD-1.1 processes execution.■ *run_idc_connmgr* is a C-shell script invoked by the Internet daemon (*inetd*) when a data provider transmits a Connection Request Frame to a data center. The script is responsible for establishing the execution environment for the *Connection Manager* and invoking that process.■ *run_idc_connsvr* is a C-shell script invoked by the Internet daemon when a *Connection Manager* process contacts one of the data center computers hosting inbound connections. The script is responsible for establishing the execution environment for the *Connection Manager Server* and invoking that process.■ *run_idc_dcmgr* is a C-shell script used to establish an execution environment and invoke a *ForeMan* process. This script is executed on each computer hosting either a *Data Parser* or a *Connection Originator* instance.

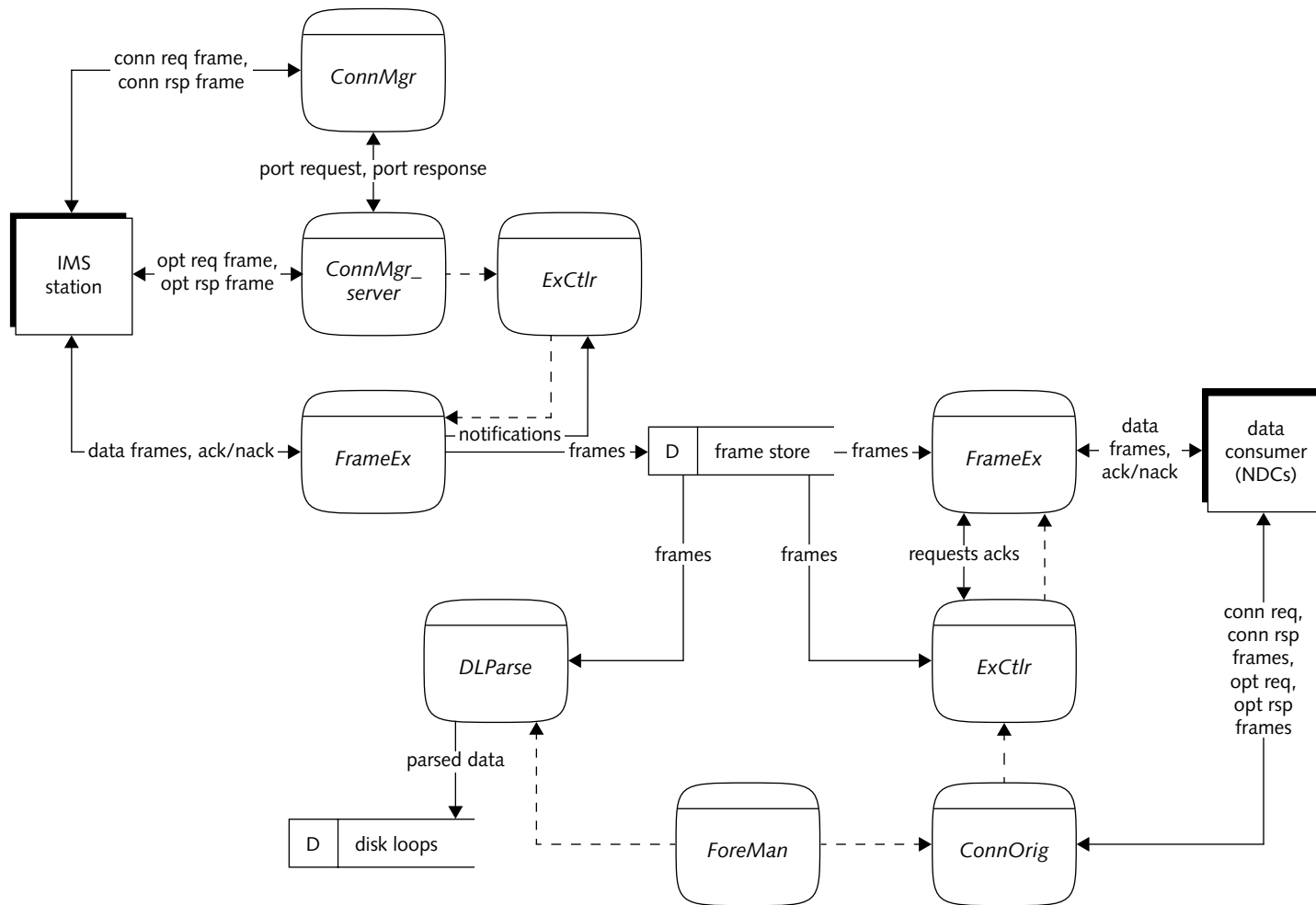


FIGURE 3. CDS CD-1.1 ARCHITECTURAL DATA FLOW

ENVIRONMENT AND STATES OF OPERATION

Software Environment

The *CDS CD-1.1* requires a minimum set of resources. The following paragraphs identify required resources:

- The *CDS CD-1.1* software is intended to run on computers with UNIX-based operating systems.
- Disk space is required to house data received via the *CDS CD-1.1* software. Twenty stations providing 3 channels of data once every 10 seconds and a 10-day Frame Store would require 10 GB of disk.
- Connections to a WAN are required by the *CDS CD-1.1* computer to receive connection requests and data and to provide data to subscribing data centers. If a network firewall exists, exceptions will need to be configured to allow inbound and outbound connections to recognized remote systems.
- The *CDS CD-1.1* software utilizes the ORACLE 8 DBMS for processing connections and handling data. The DBMS instance and table definitions are the same for the CD-1.1 software as for the CD-1.0 software.
- The *CDS CD-1.1* software is designated for execution on Sun Microsystems computers running the Solaris 2.7 variant of UNIX.

["Chapter 4: Installation Procedures" on page 105](#) provides detailed instructions for establishing a *CDS CD-1.1* software environment containing the above elements. Installation of the Solaris operating system and the ORACLE DBMS are not addressed by this document and are assumed to have been installed prior to installation or operation of the *CDS CD-1.1* software.

Normal Operational State

Two means are used to activate the *CDS CD-1.1*: automatic invocation based on operating system configuration and nonautomatic/interactive startup. The second mode may also be configured into the operating system for auto-startup if desired. After it is started, the *CDS CD-1.1* software for receiving, forwarding, and parsing data is designed to run indefinitely. The normal operational state of the software is to process protocol frames as they become available and to do so forever. See [“Operational Procedures” on page 15](#) for additional information.

Chapter 2: Operational Procedures

This chapter provides instructions for using the software and includes the following topics:

- [Software Startup](#)
- [Software Shutdown](#)
- [Maintenance](#)
- [Security](#)

Chapter 2: Operational Procedures

SOFTWARE STARTUP

The Continuous Data Subsystem for CD-1.1 requires minimum operator intervention for startup. Two categories of processes with respect to startup are those with automated invocations and those with nonautomated invocations.

Automated Invocations

After installation and configuration are complete, the execution of processes supporting inbound data are automatic. Automated inbound invocations are presented on the left side of [Figure 4](#) and described below:

1. A Connection Request Frame is sent to the “well known” host and port of the data center.
2. The *inetd* process forks the *Connection Manager* process, passing it the Connection Request Frame.
3. *Connection Manager* sends a message to a host and port requesting service of a *Connection Manager Server*.
4. *inetd* forks the *Connection Manager Server* process, passing it the message from the *Connection Manager*.
5. *Connection Manager Server* replies to *Connection Manager* with the port to use for communication.
6. *Connection Manager* makes a trial connection to the *Connection Manager Server* port.

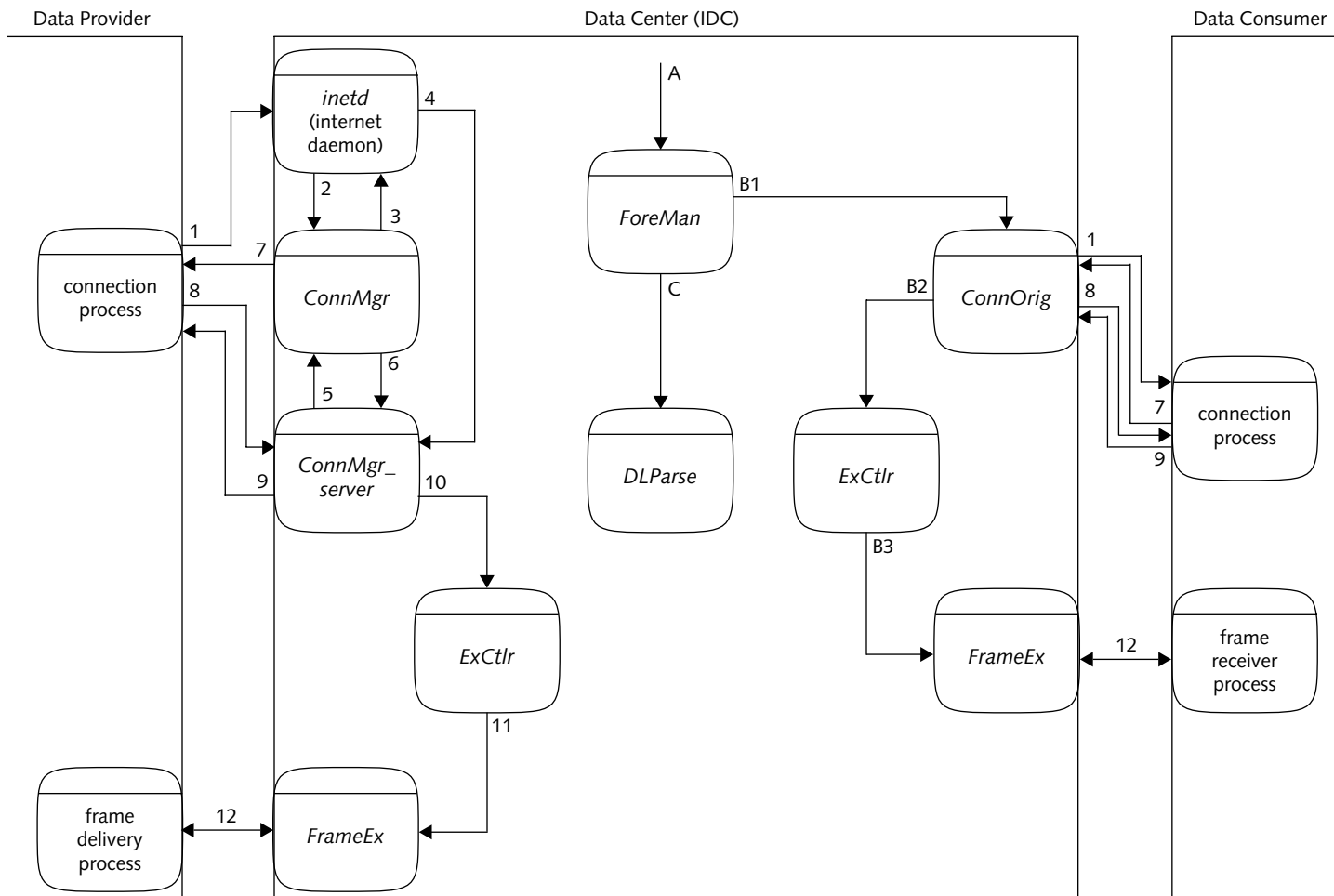


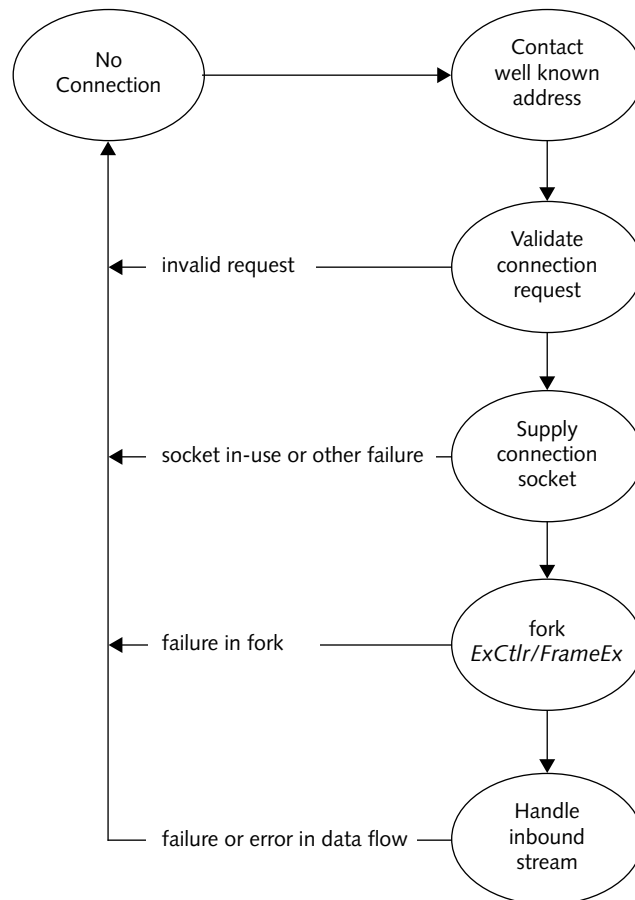
FIGURE 4. CDS CD-1.1 PROCESS INVOCATION

▼ Operational Procedures

7. *Connection Manager* replies with a *Connection Response Frame* providing the communication port.
8. An *Option Request Frame* is sent to the *Connection Manager Server* requesting a connection.
9. *Connection Manager Server* sends an *Option Response Frame* to the requestor.
10. *Connection Manager Server* forks the *Exchange Controller* and passes it an open port file descriptor.
11. *Exchange Controller* forks the *Frame Exchange* passing it the port file descriptor.
12. Frame traffic flows to the *Frame Exchange* process.

[Figure 5](#) shows a state transition of data center connection processing. The states and transitions are as follows:

- A “no connection” state leads to connection processing when the data provider contacts the “well known” address and port of the data center.
- The Internet daemon forks the *Connection Manager* process leading to the “validate connection request” state. If an invalid request was received, processing is returned to the “no connection” state. That is, no processing is active.
- A valid request leads to a state wherein a connection socket is allocated for the data provider. If an error is encountered in socket allocation, control transitions to the “no connection” state.
- A successful allocation transitions the system to the “fork *Exchange Controller/Frame Exchange*” state, where the *Frame Exchange* and *Exchange Controller* processes are started for handling the subsequent flow of frames. If an error occurs in forking these processes, control transitions to the “no connection” state.

**FIGURE 5. CONNECTION STATE TRANSITIONS**

- Successful forking of processes results in a transition to handle the inbound stream. The system remains in this state until a terminating error is encountered. In such an event, the software exits, and control transitions to the “no connection” state. From the “no connection” state, the provider participating in the protocol can/should attempt to re-establish its connection.

▼ Operational Procedures

Nonautomated Invocations

Several processes are not started automatically. Processes not started automatically are the *Data Parser*, the *Data Center Manager*, and instances of *Exchange Controller* with *Frame Exchange* for outbound data. However, of these only the *Data Center Manager* requires human attention for startup, because it is responsible for other processes. To start the *Data Center Manager* issue the following command from the UNIX command line:

```
run_idc_dcmgr <hostname>
```

When configured to manage the *Data Parser* and the *Connection Originator* processes the *Data Center Manager* will start these processes, monitor them, and restart them if they exit.

The *Data Center Manager* process does not initiate or manage processes on remote hosts. Depending on how data center resources are allocated, it may be necessary to run more than one *Data Center Manager*. In such a case, log on to the selected host computer, change to the appropriate directory, and issue the command as provided above substituting the host name as appropriate.

Nonautomated invocations are represented on the right side of [Figure 4 on page 17](#) and described below:

- A. The system operator starts the *ForeMan* process. Alternatively, the *ForeMan* may be configured to start automatically at system boot.
- B1. *ForeMan* forks the *Connection Originator* to connect to a data consumer (forwarding destination). Interaction between *Connection Originator* and a data consumer complies with CD-1.1 protocol. Frame traffic is the same as on the left side of [Figure 4 on page 17](#) between a data provider and the data center. See ["Automated Invocations" on page 16](#) for more information.
- B2. *Connection Originator* forks *Exchange Controller* and passes an open port file descriptor to the data consumer.
- B3. *Exchange Controller* forks *Frame Exchange* passing it the port file descriptor.

- C. *ForeMan* forks the *Data Parser* to process frames that are in the Frame Store.

Manual Methods

The *CDS CD-1.1* may also be started manually. These methods are not recommended for starting the software in an operational environment. This information is provided only to aid in diagnostic situations. The following assumptions are made for these instructions:

- The user is logged on as the *CDS CD-1.1* user or similar user account that has permission to execute the *CDS CD-1.1* programs and write to the Frame Store and the logging directories.
- The execution environment has been established and includes the definition of the run-time environment variable. To accomplish this execute a source `cds_idc_env` command.
- All commands are issued from the "bin" directory for *CDS CD-1.1* applications (BINroot, or <CDSroot/bin>).

Connection Manager Server

The *Connection Manager Server* supports the *Connection Manager* to service a connection request from a data provider. In a manual mode you must execute both of these processes to establish a connection:

1. Issue a command to start the *Connection Manager Server* on the host designated to service the data flow connection as follows:

```
ConnMgr_server <CONTINUOUS_DATA-DIR>/ConnMgr/connsvr_
<inst>.par
```

The process is started; but no output is directed to the screen.

2. Verify activation by checking the following log:

```
<LOGroot>connsvr_log_ <inst>.<iter>
```

▼ Operational Procedures

When the *Connection Manager Server* process terminates, the shell's interactive prompt will return only if the process was unsuccessful. If the process was successful it will have created the *Exchange Controller* process, which takes over the shell and will execute indefinitely.

Connection Manager

The *Connection Manager* and the *Connection Manager Server* operate together to service a connection request from a data provider. In a manual mode, both processes must be executed to establish a connection:

1. Ensure that the *Connection Manager Server* is running before the *Connection Manager* is started if it is possible that a connection request may be received.
2. Issue a command to start the *Connection Manager* on the host where inbound connection requests will be directed by the data provider:

```
ConnMgr <CONTINUOUS_DATA-DIR>/ConnMgr/connmgr_idc.par
```

The process is started, but no output is directed to the screen.

3. Verify activation by checking the following log file:

```
<LOGroot>connman_idc.<iter>
```

The shell's interactive prompt will not return until the program completes/terminates.

The *Connection Manager* will be suspended waiting for I/O from a data provider. Until that input is received, the process will remain active (waiting on the I/O, but doing nothing).

Connection Originator

The *Connection Originator* is used to establish a connection to a data consumer, that is, to push data from the current location to a destination location. To start the *Connection Originator*:

1. Issue the command to start the *Connection Originator* on the host configured to supply data. If that host has a firewall, it must be configured to allow the connection exception:

```
ConnOrig self=<self-name> connection=<connect-name> \  
<CONTINUOUS_DATA-DIR>/ConnOrig/ConnOrig.par
```

In this command <self-name> is the name of the local site. This name must be exactly as used to refer to the local site in defining the frame sets supporting data delivery (see ["Frame Store" on page 129](#)). The token <connect-name> is the name of the remote site and similarly must be exactly as the name used in the configuration of frame sets.

No output is directed to the screen by the *Connection Originator*.

2. Verify activation by checking the following log file:

```
<LOGroot>/ConnOrig_<self>_<connection>.<iter>
```

When the *Connection Originator* process terminates, the shell's interactive prompt will only return if the process was unsuccessful. If the process was successful, it will have created the *Exchange Controller* process, which takes over the shell and will execute indefinitely.

Exchange Controller

The *Exchange Controller* takes as inputs open file descriptors needed for communication to a protocol peer. For this reason this process cannot be started manually and must be initiated by either the *Connection Originator* or the *Connection Manager Server*.

Frame Exchange

The *Frame Exchange*, like the *Exchange Controller*, takes open input file descriptors as input. For this reason the *Frame Exchange* cannot be started manually and must be initiated by the *Exchange Controller*.

▼ Operational Procedures

Data Parser

The *Data Parser* will parse data contained in configured frame sets. Execute the *Data Parser* as follows:

1. Issue the command to start the *Data Parser* on a host that has visibility to the disk partitions containing the relevant Frame Store and Disk Loop files:

```
DLParse <CONTINUOUS_DATA-DIR>/DLParse/DLid_<inst>.par
```

No output is directed to the screen by the *Data Parser*.
2. Verify activation by checking the log file <LOGroot>/DL<inst>.<iter>.

The *Data Parser* will execute indefinitely if no problems are encountered; control will not be returned to the shell's interactive prompt. If the shell prompt is presented, the *Data Parser* has exited, typically because an error condition was encountered.

Data Center Manager

The *Data Center Manager* will start and monitor the execution of other processes that run on data center computers (see ["Nonautomated Invocations" on page 20](#)). If the configuration of the *Data Center Manager* includes definitions to start the *Data Parser* or the *Connection Originator*, and these processes have been started via the manual method, then the *Data Center Manager* should not be executed. Starting *Data Center Manager* under these conditions would cause multiple processes requesting the same resource to be started and produce an unstable environment. To execute the *Data Center Manager*:

1. Issue the command:

```
ForeMan <CONTINUOUS_DATA-DIR>/ForeMan/ForeMan_<inst>.par
```

No output is directed to the screen by the *Data Center Manager*.
2. Verify activation by checking the following log file:
<LOGroot>/foreman_<inst>.<iter>

The *Data Center Manager* will execute indefinitely if no problems are encountered; control will not be returned to the shell's interactive prompt. If the shell prompt is presented the *Data Center Manager* has exited because of an error condition.

SOFTWARE SHUTDOWN

Shutdown of *CDS CD-1.1* software, like startup, has distinct methods for automatic and nonautomatic invocations.

Shutdown for Automated Invocations

The *CDS CD-1.1* is designed to be a highly reliable and operate without human intervention. While this has benefits in ongoing operations, it makes terminating operation slightly more cumbersome. Shutdown is a two-step process. First, automated invocations must be disabled. This is necessary because when a data provider connection is broken, the provider is likely to attempt to re-establish the connection. If auto-invocation is not disabled, the *CDS CD-1.1* software will restart with a new instance to service the likely reconnect request. The second step is to terminate active *CDS CD-1.1* processes. Shut down the *CDS CD-1.1* by first disabling the automated startup of *CDS CD-1.1* components as follows:

1. Log on to the computer hosting the *Connection Manager* process that supplies the contact point for data providers.
2. Connect/change to the binary directory for *CDS CD-1.1* application software (<BINroot>, or <CMSroot>/bin) where the file `run_idc_connmgr` should be found. Rename/move this file to some other name. For example:

```
mv run_idc_connmgr run_idc_connmgr.save
```

This operation will disable the Internet daemon processing, which is configured to execute the `run_idc_connmgr` script. That is, when a connection request is received, the *inetd* will attempt to run the script, but because it has been renamed, the *file* will not be found and will fail, preventing the automated invocation and data provider from connecting.

▼ Operational Procedures

After automatic invocations are disabled, existing processes must be terminated. The following steps must be executed on each host that is servicing a data connection to be terminated. It is possible to terminate the *CDS CD-1.1* processes on a selective basis, that is, not all processes must be shut down because a particular instance of connection servicing processes is to be terminated.

1. Log on to the host (or hosts) that services connections for the data provider(s). This will be the host for the *Connection Manager Server* process(es).
2. Issue a UNIX process status command to find the *CDS CD-1.1* processes that are active/running. For example:

```
ps -fu <cds-user-name>
```

This command will produce output similar to the following:

UID	PID	PPID	C	STIME	TTY	TIME	CMD
cmss	7350	7348	0	May 22	pts/10	0:01	-csh
cmss	8402	8401	0	May 22	?	18:56	Frame-Exchange par=/ data/spinach/cds11/config/app_config/continuous_data/ExCtrlr/ctrlr_fex_idc_CD01.par SOCKFD=
cmss	6858	1	0	May 24	?	0:01	/data/spinach/cds11/bin/ExCtrlr connection=CD02 par=/data/spinach/cds11/c
cmss	8401	1	0	May 22	?	0:01	/data/spinach/cds11/bin/ExCtrlr connection=CD01 par=/data/spinach/cds11/c
cmss	6859	6858	0	May 24	?	53:47	Frame-Exchange par=/data/spinach/cds11/config/app_config/continuous_data/ExCtrlr/ctrlr_fex_idc_CD02.par SOCKFD=

3. The *Frame Exchange* and *Exchange Controller* support *CDS CD-1.1* data flow. Issue a `kill` command to the process identification(s) (PID) of the *Exchange Controller(s)* to shut down the data flow (the *CDS CD-1.1* support). For example, using the output in the example shown in step 2 above:

```
kill 6858 8401
```

4. After issuing the `kill` command, the software will begin cleanup and shutdown processing. After a few moments issue another process status command to determine if the processes are still active/present in the process list.
5. If after four minutes the processes are still listed in the process list with the same PIDs, as before, use a more aggressive version of the `kill` command:

```
kill -9 6858 8401
```

This version of the command will forcibly terminate outstanding I/O and terminate the process.

When the *CDS CD-1.1* is to be reactivated, rename the `run_idc_connmgr` script to its proper name (see [step 2 on page 25](#)). No other action is required to restore automated connection processing.

Shutdown for Nonautomated Invocations

Shut down the nonautomated invocations by terminating the controlling *CDS CD-1.1* process:

1. Log on to the computer hosting the processes to be terminated. Nonautomated invocation processes are executed on machines where a *Data Center Manager* process has been configured to execute.
2. Use a process status command to get a list of the PIDs of the *CDS CD-1.1* processes. For example:

```
ps -fu <cds-user-name>
```

This command will produce output similar to the following:

UID	PID	PPID	C	STIME	TTY	TIME	CMD
cmss	7350	7348	0	May 22	pts/10	0:01	-csh
cmss	240	237	0	May 28	?	1:04	/data/spinach/ cds11/bin/DLParse par=/data/spinach/cds11/config/ app_config/continuous_data/DLParse/DLi

▼ Operational Procedures

```

cmss  28230  8208  0    May 22 ?      303:59 Frame-
      Exchange/par=data/spinach/cds11/config/app_config/
      continous_data/ExCtrlr/ctrlr_fex_IDC_NDC.par SOCKFD=5
cmss  237  26922  0    May 28 ?      0:00 /bin/csh /data/
      spinach/cds11/bin/run_idc_dcmgr
cmss  26922  1  0    May 22 ?      0:17 ForeMan
      par=ForeMan_idc.par
cmss  28208  26922  0    May 22 ?      0:38 /data/spinach/
      cds11/bin/ExCtrlrSOCKFD=5par=/data/spinach/cds11/config/
      app_config/
      continous_dataExCt
cmss  28729  28727  0    May 30 pts/6    0:00 -csh

```

3. Issue a `kill` command to the PID of the *ForeMan* (Data Center Manager) process using the output in the example in step 2 above:

```
kill 26922
```

4. After issuing the `kill` command, the software will begin cleanup and shutdown processing. The *ForeMan* communicates with all its managed processes to tell them to shut down. After a few moments, issue another process status command to determine if the processes are still active/present in the process list. In the above example, the `kill` would result in terminating the following processes:

```

Data Parser:      PID 240
Frame Exchange:   PID 28230 is supporting a forwarding connection
                  to a NDC
Exchange Controller: PID 28208 is the companion to the Frame
                  Exchange for the forwarding connection
run_idc_dcmgr:    PID 237 script that runs the Data Center Manager

```

5. If after four minutes the processes are still listed in the process list with the same PIDs, then use a more aggressive version of the `kill` command as follows:

```
kill -9 26922
```

This version of the command will forcibly terminate outstanding I/O and terminate the process.

When the processing on the terminated host is to be resumed, issue a command to run the *Data Center Manager* run script. For example:

```
run_idc_dcmgr <host-name>
```

MAINTENANCE

After the CDS *software* is configured for operation with a set of protocol peers, it requires no routine intervention by the human operator.

To add a reporting site to the operational configuration see ["Adding Data Providers" on page 140](#). To add a forwarding destination see ["Adding Forwarding Destination" on page 155](#).

This section describes how the CDS *CD-1.1* handles files written. All the CDS *CD-1.1* applications write to log files to provide information about processing and error conditions. These log files are managed by library software within each process and have the following attributes:

- The number, size, path, and name of log files are all configurable.
- Output is directed to the log file until the configured size of the file is exceeded. The file is then renamed and a new iteration of the file is created. Output is then directed to this new iteration of the log file.
- The oldest iteration version is deleted when the configured number of log files is reached.

As a result of these attributes there is no need for operator intervention to maintain or remove log files.

Other files written by the CDS *CD-1.1* software include Frame Store files and frame log files. Both of these classes of files are managed by library software within the Continuous Data Subsystem for CD-1.1. Through run-time configuration parameters the library software controls the size and the number of files maintained. The frame log files are treated like rolling windows, that is, after the log is filled, the oldest portions are removed/truncated as newer portions are written. Frame Store files are handled like log files, where each file is filled a new one is created. When

▼ Operational Procedures

the configured number of files is reached (actually expressed as a time duration in the configuration) the oldest file is deleted. In this way the whole frame-set directory's content is like the rolling window of the frame log. As with log files, these management strategies require no operator intervention during normal operation.

SECURITY

Security for general operation of *CDS CD-1.1* software is provided by conventional UNIX user accounts, group membership, file permissions, and passwords. No special security requirements are required for a *CDS CD-1.1* user account. A standard user account is created by a system administrator with no further modification. If installation requires setting access restrictions to the *CDS CD-1.1* executables and files or restricting the capabilities of the *CDS CD-1.1* user (account), the UNIX security controls described above may be used.

The *Data Parser* and *Connection Manager* components of the *CDS CD-1.1* use the monitoring system DBMS. Access to the DBMS is controlled via a user-name/password pair to a specified account of the DBMS. This information is specified as part of the run-time configuration of each of these processes. See ["Chapter 4: Installation Procedures" on page 105](#) for each of these components for additional information.

Security for protocol connections and frames is provided by digital signatures contained in the protocol frames. Signatures are created and verified with private and public keys, respectively. Keys for processing signatures must be provided in the configuration environment to establish connections and verify frames as being "untampered" after the connection is established. See ["Authentication" on page 134](#) for additional information.

Chapter 3: Troubleshooting

This chapter describes how to identify and correct problems related to the *CDS CD-1.1* and includes the following topics:

- [Monitoring](#)
- [Interpreting Error Messages](#)
- [Solving Common Problems](#)
- [Reporting Problems](#)

Chapter 3: Troubleshooting

MONITORING

CDS CD-1.1 programs are monitored through the use of log files written by the various components and the analysis of operating system process lists. Log files are written to a configurable directory by each CD-1.1 process. The names of the log files are also configurable and typically represent the instance of the process writing the file. For example, *Data Parser* with identifier 100 will write the log file `DL100.log`. The *Frame Exchange* supporting the forwarding connection between the IDC and NDC-BB will write the log file `fex_IDC_NDC-BB.log`. Because the names are configurable, there is a risk of creating ambiguities and confusion. This should be avoided by selecting appropriate log file names.

Monitoring the *CDS CD-1.1* is recommended as a once daily activity when the sub-system is operational in a steady-state environment. During initial setup, or when data providers are being added or removed, the system should be monitored more frequently to ensure timely identification of any problems introduced by changes in the system.

Inbound Data and Connections

To check for processes supporting an inbound connection, issue the following command at the UNIX command line:

```
ps -fu <cds-user-name>
```

This process status (`ps`) command sets options for full (`x`) information on processes owned by user (`u`) `<cds-user-name>`. For example, `cmss` is the name of the user account from which *CDS CD-1.1* processes are executed. This command produces output similar to the following:

```

UID    PID    PPID    C    STIME TTY      TIME CMD
cmss   7350   7348    0    May 22 pts/10   0:01 -csh
cmss   8402   8401    0    May 22 ?       18:56 Frame-Exchange par=/data/
        spinach/cds11/config/app_config/continuous_data/ExCtlr/
        ctrl_fex_idc_CD01.par SOCKFD=
cmss   6858    1    0    May 24 ?       0:01 /data/spinach/cds11/bin/
        ExCtlr connection=CD02 par=/data/spinach/cds11/c
cmss   8401    1    0    May 22 ?       0:01 /data/spinach/cds11/bin/
        ExCtlr connection=CD01 par=/data/spinach/cds11/c
cmss   6859   6858    0    May 24 ?       53:47 Frame-Exchange par=/data/
        spinach/cds11/config/app_config/continuous_data/ExCtlr/
        ctrl_fex_idc_CD02.par SOCKFD=

```

This output shows two *Exchange Controller* processes and two *Frame Exchange* processes. The process status shows that PID 8401 is an *Exchange Controller* servicing the connection to site CD01 (the connection parameter on the status line is set to CD01). PID 8402 is a *Frame Exchange* servicing the inbound connection from site CD01 (the par file being used by that process, `ctrl_fex_idc_CD01.par`, specifies its use). Similar logic shows that the other exchange/controller pair supports a connection to site CD02.

The `ps` command shows that the processes are active as far as the operating system is concerned. For more information about these processes, examine the log files. The `ps` output does not list *Connection Manager* or *Connection Manager Server* processes. This is to be expected; when these processes execute successfully they are only active for a short time. If for some reason successive `ps` commands (over a period of several minutes) show the same *Connection Manager* or *Connection Manager Server* process(es) as active (PIDs must be the same in successive `ps`), then something is wrong. When the *Connect Manager* or *Connection Manager Server* have not terminated, the operator should issue a terminate signal to the process with a UNIX `kill` command and investigate, or vice versa.

Log files contain information written by the application software. The amount of information written is configurable in the par files for the various components. In general, the log level should be set relatively low (values range from 1–10) so that

▼ Troubleshooting

only errors and critical information are logged. When the log level is set higher it is sometimes difficult to quickly ascertain how/what a process is doing. Log level 5 is recommended for steady-state operations.

To further check on the processes supporting the inbound data flow addressed above, change to the *CDS CD-1.1* log directory. A list command (ls) in the log directory should produce a listing similar to the following:

```
connman_idc.01      ctrlr_idc_CD01.03 ctrlr_idc_CD02.04 flex_idc_CD01.05
connman_idc.02      ctrlr_idc_CD01.04 ctrlr_idc_CD02.0  flex_idc_CD02.01
connsvr_log_idc1.01 ctrlr_idc_CD01.05 flex_idc_CD01.01 flex_idc_CD02.02
connsvr_log_idc1.02 ctrlr_idc_CD02.01 flex_idc_CD01.02 flex_idc_CD02.03
ctrlr_idc_CD01.01   ctrlr_idc_CD02.02 flex_idc_CD01.0  flex_idc_CD02.04
ctrlr_idc_CD01.02   ctrlr_idc_CD02.03 flex_idc_CD01.0  flex_idc_CD02.05
```

This listing shows two *Connection Manager* log files (*connman_idc*), one for each of the inbound connections being serviced. There are also two *Connection Manager Server* files (*connsvr_log_idc*), one for each connection request. The remaining log files are for each instance of the *Exchange Controller* and *Frame Exchange*.

The contents of the *connman_idc.<xx>* file should provide evidence of the connection request and should terminate with the creation of a Connection Response Frame, as follows:

```
Program started on Wed May 24 15:45:54 2000
=====
[info]: Parameter use_database = yes
[info]: Parameter inetd = 1
[info]: Parameter framestore-file = /data/spinach/cds11/bin/
        FrameStore/fstore_idc.par
[info]: Parameter current-location = IDC
[info]: Open Database cd11test/cd11@pumpkin
[info]: Database Opened
[info]: Read Frame Size = 84
.
.
.
[info]: Received connection request frame from CD02
[info]: Checked station CD02 address 140.162.5.5 in database
[info]: Selected 2 running servers from DLMAN
[info]: Reorder Servers
[info]: Reordered array of servers!
[info]: Lock station CD02 in table ALPHASITE
```

```

[info]: Locked station CD02 in table ALPHASITE
[info]: Check server DLID = 100 address spinach port 8032
[info]: Connected to address spinach port 8032
[info]: Sent port request for station CD02
[info]: Received port response 8034
[info]: Connected to address spinach port 8034
[info]: Checked address spinach port 8034
[info]: Update station CD02 table ALPHASITE
[info]: Updated station CD02 address 140.162.5.17 time 959183155 dlid
      100 in db
[info]: fd1 closed
.
.
.
[cdo_build]: Built frame length 84
[info]: Created connection response frame size = 84
[info]: Wrote frame to socket size = 84
[info]: Database Closed
=====
Program exited on Wed May 24 15:45:55 2000

```

Similarly, the contents of the `connsvr_log_idc.<xx>` should show evidence of the processing of a connection request. This file should terminate with lines that indicate the *Frame Exchange* (actually the *Exchange Controller*) has been started, similar to the following:

```

Program started on Wed May 24 15:45:55 2000
=====
[info]: Parameter inetd = 1
[info]: Received port request for station CD02
[info]: Bound on 8034 port!
[info]: Listen on spinach.cmr.gov port 8034
[info]: Sent port response for station CD02 port 8034
[info]: Accepted connection on a socket
[info]: Accepted connection on a socket
[info]: Read Frame Size = 72
.
.
.
[info]: Received option request frame from CD02
[cdo_build]: ==Build frame type: 4, creator
[cdo_pack]: Call pack segment for frame type 4, creator
.
.
.

```

▼ Troubleshooting

```
[cdo_build]: Built frame length 64
[info]: Size of OAF frame built size = 64
[info]: Wrote frame to socket size = 64
[info]: Sent Option Response Frame!
[info]: Starting frame exchange...
[info]: fd1 closed
[info]: fd2 closed
=====
Program exited on Wed May 24 15:45:55 2000
```

With the log level set to announce only error conditions (log level 3), the log file of the *Frame Exchange* (*fex_idc_<inst>.<xx>*) should be sparse, as shown:

```
Program started on Mon May 22 14:29:24 2000
=====
```

The log level of the *Exchange Controller* set at the information level (6) will result in logging of all frame transactions. Using the UNIX `tail -f` command on the most recent controller log file (the one with a `.01` suffix) will display the progress of inbound frame delivery. Example output is as follows:

```
[info]: process_rcved_frame   received CDO_FT_DAT frame SSN: 94008
[info]: run_controller_exec:  At the top of the controller run loop
[info]: process_rcved_frame:  received CDO_FT_DAT frame SSN: 94009
[info]: run_controller_exec:  At the top of the controller run loop
[info]: run_controller_exec:  At the top of the controller run loop
[info]: log_heart_beat:       Creating Heart Beat
[info]: run_controller_exec:  At the top of the controller run loop
[info]: process_rcved_frame:  received CDO_FT_DAT frame SSN: 94010
```

Similar output should continue as long as the connection is open and the provider is delivering frames. When using the `tail` command be aware that log files are closed after attaining a configured size. A new file of the same name is created, and the old one is renamed. If a `tail -f` is left executing, the output will eventually indicate that the file has been closed and will cease being updated. Reissuing the `tail` command will resume the monitoring of the (new) file. It is possible for the connection to be open, but have no frames received; this is not an error condition for the receiver and may not be an error for the sender either. It means that no frames are being sent by the provider. The connection should remain open, and when the provider begins sending frames, the announcement of received frames

will begin to show up in the *Exchange Controller's* log file. When no frames are being received, the log file will contain a succession of "at the top of the controller run loop" and "heart beat entries." If the log file is full of these entries, a `ps` command will verify that the *Exchange Controller's* companion *Frame Exchange* process is still active. If the log file is not full of these entries, the `kill` command should be used to send a terminate signal to the *Exchange Controller*. Until the *Exchange Controller* is terminated the provider will be unable to reconnect and restart the flow of frames. This condition should not occur because both the *Exchange Controller* and *Frame Exchange* are designed to exit if their companion process is not active.

Processes Controlled by Data Center Manager

The same `ps` command used for inbound support processes can be used to monitor those processes managed directly or indirectly by the *Data Center Manager* (*ForeMan*):

- *Connection Originator* (*ConnOrig*)
- *Data Parser* (*DLParse*)
- *Frame Exchange* (*FrameEx*) used for forwarding connections
- *Exchange Controller* (*ExCtrlr*) used for forwarding connections

All of these processes are directly or indirectly started and restarted by the *ForeMan* process. After *ForeMan* has been run and the software processes have stabilized (a couple of minutes), issue the `ps` command:

```
ps -fu <user-name>
```

Output similar to the following will be produced:

```

UID      PID  PPID  C    STIME TTY      TIME CMD
cmss    7350  7348  0    May 22 pts/10   0:01 -csh
cmss    240   237  0    May 28 ?        1:04 /data/spinach/cds11/bin/
        DLParse par=/data/spinach/cds11/config/app_config/
        continuous_data/DLParse/DLi
cmss 28230 28208  0    May 22 ?        303:59 Frame-Exchange par=/data/
        spinach/cds11/config/app_config/continous_data/ExCtrlr/
        ctrlr_fex_IDC_NDC.par SOCKFD=5
```

▼ Troubleshooting

```

cmss      237 26922  0   May 28 ?           0:00 /bin/csh /data/spinach/
          cds11/bin/run_idc_dcmgr
cmss      26922    1  0   May 22 ?           0:17 ForeMan
          par=ForeMan_idc.par
cmss      28208 26922  0   May 22 ?           0:38 /data/spinach/cds11/bin/
          ExCtlr SOCKFD=5 par=/data/spinach/cds11/config/app_config/
          continuous_data/ExCt
cmss      28729 28727  0   May 30 pts/6       0:00 -csh

```

The output shows *Exchange Controller* and *Frame Exchange* processes. The process status shows that PID 28230 is a *Frame Exchange* servicing the outbound connection to the NDC site (the par file being used by that process `ctlr_fex_IDC_NDC.par` specifies its use). The *Exchange Controller* servicing the connection to the NDC site is inferred because the architecture dictates that there must be a controller for each *Frame Exchange*. If there were other *Exchange Controllers* active in support of inbound connections, their use would be indicated by command parameters (as described in the previous paragraphs). The process list also shows active *ForeMan*, *run_idc_dcmgr*, and *Data Parser* processes. The *run_idc_dcmgr* process is a script that invokes *ForeMan*. If one of these processes is present without the other, a problem is indicated. If the *Data Parser* or the exchange/controller pair supporting the outbound connection(s) is not present in the process list, but the *ForeMan* is still active, the *ForeMan* should restart the missing process after the configured amount of wait time. The `ps` command shows that the processes are active as far as the operating system is concerned. For information about these processes, examine the log files they write.

The `ps` output may not show *Connection Originator* processes because these processes are only active for a short time. If `ps` commands (over several minutes) show the same *Connection Originator* process, then something is wrong. If there are problems establishing a connection, the *ForeMan* will keep restarting the *Connection Originator* to attempt the connection. Therefore, it is important to notice whether the PID of the *Connection Originator* is changing from one `ps` to another. If the *Connection Originator* is hung-up, the operator should issue a terminate signal to the process with a UNIX `kill` command and investigate, or vice versa.

To check the log of the data center processes addressed above, `cd` to the *CDS CD-1.1* log directory. A `ls` command in the log directory should produce a listing similar to the following:

```
ConnOrig_IDC.01   ctrlr_idc_NDC1.05   fex_idc_NDC1.01   foreman_IDC.02
ConnOrig_IDC.02   DL100.log.01       fex_idc_NDC1.02   foreman_IDC.03
ctrlr_idc_NDC1.01 DL100.log.02       fex_idc_NDC1.03   foreman_IDC.04
ctrlr_idc_NDC1.02 DL100.log.03       fex_idc_NDC1.04   foreman_IDC.05
ctrlr_idc_NDC1.03 DL100.log.04       fex_idc_NDC1.05
ctrlr_idc_NDC1.04 DL100.log.05       foreman_IDC.01
```

The contents of the `ConnOrig_IDC.<xx>` file should provide evidence of the connection request and terminate with lines indicating that the *Exchange Controller* has been started, similar to the following:

```
=====
ConnOrig[info]: print_hello_message : 0 : This is Connection Originator version '%Y%', built '000512 09:09:05 PDT', running at '20000522 21 : 10:50'
ConnOrig[info]: co_portinfo_get : 0 : will try 1 IP addresses
ConnOrig[info]: try_one_portrange : 0 : trying to connect at onion.cmr.gov, ports 8031-8031
ConnOrig[info]: open_stream_socket : 0 : opened stream socket 5
ConnOrig[info]: timed_connect : 0 : connecting to address 140.162.5.8, port 8031
ConnOrig[info]: timed_connect : 0 : connect to socket 5 succeeded
ConnOrig[info]: co_frames_pack_crf : 0 : built connection request frame (84 bytes)
ConnOrig[info]: send_frame : 0 : sending frame (84 bytes) ...
ConnOrig[info]: send_frame : 0 : sent frame (84 bytes)
ConnOrig[info]: recv_frame : 0 : receiving a frame on socket 5...
ConnOrig[info]: recv_frame : 0 : received frame (84 bytes)
ConnOrig[info]: co_frames_unpack_crs : 0 : unpacked crs, 84 bytes
ConnOrig[info]: close_tcp_connection : 0 : closing socket 5
ConnOrig[info]: open_stream_socket : 0 : opened stream socket 5
ConnOrig[info]: timed_connect : 0 : connecting to address 140.162.5.8, port 8033
ConnOrig[info]: timed_connect : 0 : connect to socket 5 succeeded
ConnOrig[info]: co_frames_pack_orf : 0 : built option request frame (68 bytes)
ConnOrig[info]: send_frame : 0 : sending frame (68 bytes) ...
ConnOrig[info]: send_frame : 0 : sent frame (68 bytes)
```

▼ Troubleshooting

```

ConnOrig[info]: recv_frame : 0 : receiving a frame on socket 5...
ConnOrig[info]: recv_frame : 0 : received frame (64 bytes)
ConnOrig[info]: unpack_oaf : 0 : unpacked oaf, 64 bytes
ConnOrig[info]: log_execed_command : 0 : calling execv with the
        following arguments:
ConnOrig[info]: log_execed_command : 0 :      argv[0] : /data/spin-
        ach/cds11/bin/ExCtrlr/ExCtrlr
ConnOrig[info]: log_execed_command : 0 :      argv[1] : SOCKFD=5
ConnOrig[info]: log_execed_command : 0 :      argv[2] : par=/data/
        spinach/cds11/bin/ExCtrlr/ctrlr_fex_IDC_NDC.par

```

The *Frame Exchange(s)* supporting outbound data flows are essentially the same as those on the inbound side. The log file should contain only error information and therefore should be rather sparse (see [“Inbound Data and Connections” on page 32](#)” as an example). Similarly, the outbound *Exchange Controller's* log file is much like the inbound side except that the log lines announce the sending of frames as opposed to the receipt of them. The following is an example of an *Exchange Controller* log file:

```

[info]: log_frame_acknowledge: Received request to log ack for SSN:
        95423
[info]: run_controller_exec: At the top of the controller run loop
[info]: get_new_frames : For set# CD01:0, set query time is:
        959985909.888862
[info]: Wow, I got 1 new frames to send
[info]: Now I'm sending frame SSN: 95426 from set: CD01:0 and size:
        21132
[info]: get_new_frames : For set# CD02:0, set query time is:
        959985914.769990
[info]: Wow, I got 1 new frames to send
[info]: Now I'm sending frame SSN: 91019 from set: CD02:0 and size:
        15872
[info]: log_heart_beat: Creating Heart Beat
[info]: run_controller_exec: At the top of the controller run loop
[info]: get_new_frames : For set# CD01:0, set query time is:
        959985924.972149
[info]: get_new_frames : For set# CD02:0, set query time is:
        959985926.922913
[info]: Wow, I got 3 new frames to send
[info]: Now I'm sending frame SSN: 91020 from set: CD02:0 and size:
        5348

```

```
[info]: Now I'm sending frame SSN: 91021 from set: CD02:0 and size:
15872
[info]: Now I'm sending frame SSN: 91022 from set: CD02:0 and size:
5348
[info]: run_controller_exec: At the top of the controller run loop
[info]: get_new_frames : For set# CD01:0, set query time is:
959985924.972149
[info]: Wow, I got 1 new frames to send
[info]: Now I'm sending frame SSN: 95427 from set: CD01:0 and size:
21132
[info]: get_new_frames : For set# CD02:0, set query time is:
959985932.429094
```

Use a `tail -f` command on the *Exchange Controller's* log file to watch the progress of frame forwarding activity. As frames move through the system, they may be monitored via the sequence number in the log entries. The inbound controller's log file will record the sequence numbers of frames received, and the outbound controller's log file will record those sequence numbers/frames forwarded.

The log file of the *Data Parser* can be verbose. When the *Data Parser* is operating correctly it will log frame read, the parsing and processing of each Channel Subframe, and the update of the database and waveform files. The following is an example of a *Data Parser* log:

```
[main]: **** FrameRead successful ****
[main]: Frame: 1 Handle: Volid: 0, SetId: 1, File: 959558400, hoff:
0, roff: 30700, size: 21132
[main]: Frame: id: 52593, type: 5, time: 959558416.202, duration:
10.000 s, logtime: 959558451.715, size=21196
[main]: Read frame of type: 5, length: 10000, time: 2000150
00:00:16.202
[main]: Processing CD01: uc48/shz
[main]: Subframe: 0, 2000150 00:00:16.203, dur: 10.000, ns: 400, sz:
1600, uc48/shz/
[main]: In wf_convert
[main]: In wf_convert_aux
[main]: In write_data, CD01, uc48/shz channel, 400 samps, 1200 bytes
[main]: writing data for channel uc48/shz
[main]: In append_wf, time 959558416.203000 old end 959558416.178000
[main]: In chkopen, /data/spinach/cds11/RunDir/idc/DL/CD01/uc48shz/
001.w
```

▼ Troubleshooting

```
[main]: In update_wf_dbbuf, nupd = 0, wfid = 49922640
[main]: In chkfull
[main]: write successful for uc48/shz, time: 959558416.203, nsamp:
      400, size: 1600
[main]: Processing CD01: uc48/shn
[main]: Subframe: 1, 2000150 00:00:16.203, dur: 10.000, ns: 400, sz:
      1600, uc48/shn/
.
.
.
```

This type of output will be continually logged to the *Data Parser's* log file as long as new frames are arriving at the data center. The `tail -f` command may be used to monitor the *Data Parser's* activity.

The *Data Center Manager (ForeMan)* receives input from the standard output (`stdout`) and standard error (`stderr`) of the processes that it starts and manages. The inputs from these child processes, as well as *ForeMan's* own log information, are written to the *ForeMan* log file. The following example shows a section of the *ForeMan* log file:

```
[childout][evh]read_and_log_fd_input :child sends 'Frame: 0 Handle:
      Valid: 0, SetId: 1, File: 959026200, hoff: 0, roff: 1238872,
      size: 21132'
[event][fdw]evq_enqueue_event :enqueueing type 4 event
[event][evh]handle_event :handling type 4 event
[childout][evh]read_and_log_fd_input :child sends 'Frame: 0 Handle:
      Valid: 0, SetId: 1, File: 959026200, hoff: 0, roff: 1260068,
      size: 21132'
[info][rpr]rpr_reaper_main_loop :process 26930 reaped
[event][rpr]evq_enqueue_event :enqueueing type 5 event
[event][evh]handle_event :handling type 5 event
[event][evh]slp_schedule_wakeup :scheduling type 3 event in
      10.000000 seconds
[event][slp]evq_enqueue_event :enqueueing type 3 event
[event][evh]handle_event :handling type 3 event
[info][evh]child_runjob_event :argv0 == '/data/spinach/cds11/bin/
      DLParse/run_data_parser'
[info][evh]child_runjob_event :argv[0] == '/data/spinach/cds11/bin/
      DLParse/run_data_parser'
[info][rpr]rpr_reaper_main_loop :process 4430 reaped
[event][rpr]evq_enqueue_event :enqueueing type 5 event
```

```
[event][evh]handle_event :handling type 5 event
[event][evh]slp_schedule_wakeup :scheduling type 3 event in
    10.000000 seconds
[event][slp]evq_enqueue_event :enqueueing type 3 event
[event][evh]handle_event :handling type 3 event
[info][evh]child_runjob_event :argv0 == '/data/spinach/cds11/bin/
    DLParse/run_data_parser'
[info][evh]child_runjob_event :argv[0] == '/data/spinach/cds11/bin/
    DLParse/run_data_parser'
```

In this log segment, the lines that begin with the string `[childout][evh]` represent an output from the `stdout` of a child process. Depending on its content, such output may or may not be an indication of a problem. In the above example, a Frame Store request was attempting to access data that had already rolled out of the active Frame Store window. This could imply that the Frame Store is configured for too short a time period, or perhaps a startup request attempted to recover a frame that correctly had rolled out of activity. The lines following the `[childout]` lines log the restarting of the *Data Parser*. Typically, the *Data Parser* should not exit and need to be restarted, so additional investigation would be necessary if such logging messages were found.

INTERPRETING ERROR MESSAGES

This section identifies fatal error messages that may be produced by the various components of the *CDS CD-1.1* software.

In some cases a recommendation is provided to check/alter a par file parameter. Unless otherwise noted, the par file referenced is the par file of the component. For example, in the *Connection Manager* section the par file would be the *Connection Manager's* par file `connmgr_idc.par`.

Connection Manager

For messages that begin with either a single word or words connected by an underbar (`_`) and followed immediately by a colon (`:`), the leading word(s) is the name of the function that produced the message.

▼ Troubleshooting

Message: Must have at least two arguments.

Description: An incorrect invocation of the process was received. The needed configuration arguments were not provided on the command line.

Action: Use the correct command line for startup.

Message: ConnMgr: Error initializing log file!

Description: Unable to create or write to the log file.

Action: Verify the log directory specified in the configuration exists and is writable by the user.

Message: initialize: Error initialize_parameters!

Description: Unable to read or parse the configuration parameters from the par file.

Action: Check the format and syntax of the par file variable definitions. There may be another adjacent log message that specifies which parameter was problematic.

Message: initialize_parameters: Error reading
 well_known_port parameter!

Description: The required parameter was not defined or was incorrectly defined.

Action: Verify the definition of *well_known_port* in the par file.

Message: initialize_parameters: Error reading database
 parameter!

Description: The required parameter was not defined or was incorrectly defined.

Action: Verify the definition of *database* parameter in the par file.

Message: `initialize_parameters: Error inetd parameter must be 0 or 1.`

Description: A configuration parameter was improperly defined.

Action: Edit the par file to provide an acceptable parameter value (0 or 1).

Message: `initialize_parameters: Error reading framestore-file parameter!`

Description: The required configuration parameter was not defined or was incorrectly defined.

Action: Verify the definition of the *framestore-file* parameter in the par file.

Message: `initialize_parameters: Error reading current-location parameter!`

Description: The required configuration parameter was not defined or was incorrectly defined.

Action: Verify the definition of the *current-location* parameter in the par file.

Message: `initialize: Error initialize_signal_handler!`

Description: An error was encountered in establishing a system signal handler.

Action: See the system administrator. There may be another adjacent log message that will provide additional information.

Message: `initialize: Error initialize_database!`

Description: An error was encountered in opening the DBMS.

Action: Check database user-name/password and instance specification provided in the configuration file. There may be another adjacent log message that will provide additional information.

▼ Troubleshooting

Message: `initialize: Error initialize_authentication!`

Description: An error was encountered in the initialization of the authentication software.

Action: Check pathname to the authentication directory hierarchy provided in the configuration file. There may be another adjacent log message that will provide additional information.

Message: `SIGPIPE signal received`

Description: A terminating signal was received from the operating system.

Action: None. The *Connection Manager* will be restarted by the Internet daemon at the next connection request. If the problem reoccurs, contact the system administrator.

Message: `SIGTERM signal received`

Description: A terminating signal was received from the operating system.

Action: None. The *Connection Manager* will be restarted by the Internet daemon at the next connection request. If the problem reoccurs, contact the system administrator.

Message: `SIGALRM signal received`

Description: A terminating signal was received from the operating system.

Action: None. The *Connection Manager* will be restarted by the Internet daemon at the next connection request. If the problem reoccurs, contact the system administrator.

Message: Error signal <sig-num> received

Description: A terminating signal was received from the operating system.

Action: None. The *Connection Manager* will be restarted by the Internet daemon at the next connection request. If the problem reoccurs, contact the system administrator.

Message: initialize_authentication: Error
process_logging_ filedes!

Description: Unable to assign file descriptor for logging authentication messages.

Action: Check definitions in the configuration files definition location and the name of the log files.

Message: process_logging_ filedes: Error fdopen to set AS
logging

Description: An invalid file descriptor was received for logging file access.

Action: Check definitions in the configuration files. The error should not occur and may indicate a coding error.

Message: process_connection_requests: Error in
receive_conn_request!
receive_conn_request: Error during
accept_connection!
accept_connection: Error during bind_to_port!

Description: An error processing result was received from the *bind_to_port* routine.

Action: Check the error log. An adjacent log message will provide additional information.

▼ Troubleshooting

Message: `bind_to_port: Invalid port number!`

Description: An unacceptable socket port number was received.

Action: The system configuration is incompatible with system operation. Check the port value provided in the par file.

Message: `bind_to_port: Error open tcp socket errno = <errno>!`

Description: The operating system returned an error on open call.

Action: Check configuration in the par file(s) for the multiple identification of the same port; check the system configuration; advise the system administrator.

Message: `bind_to_port: Set sockopt failed errno <errno>.`

Description: The operating system returned an error on the `setsockopt` call to allow reuse of the socket.

Action: See the system administrator.

Message: `bind_to_port: Error get_ip_address!`

Description: An error processing result was received from the `get_ip_address` routine.

Action: Check the error log. An adjacent log message will provide additional information.

Message: `get_ip_address: Invalid address name!`

Description: An invalid name was received by this routine.

Action: The error should not occur. The calling routine provided unacceptable input. Determine the caller and alter coding.

Message: `get_ip_address: Error gethostname!`

Description: The system was unable to find IP address of the named node.

Action: See the system administrator; the node is not registered in the operating system (for example, in the `/etc/hosts` register).

Message: `get_ip_address: Error address list null!`

Description: The system returned a null address list for the named IP node.

Action: See the system administrator. The node address is not registered in the operating system.

Message: `bind_to_port: Error get_my_address!`

Description: An error processing result was received from the `get_my_address` routine.

Action: Check the error log. An adjacent log message will provide additional information.

Message: `bind_to_port: Bind failed address <addr>
port <port-num> errno <errno>`

Description: A failure was encountered in establishing a socket connection; *errno* identifies the returned system error.

Action: See adjacent log message(s) for additional information. The system configuration is problematic. Check `par` file definitions, and check with the system administrator for available port numbers and system names.

▼ Troubleshooting

Message: `accept_connection: Error during accept
 errno = <errno>!`

Description: A failure was encountered in accepting a connection from the requesting node; *errno* provides the system error number.

Action: See adjacent log message(s) for additional information. The system configuration is problematic. Check par file definitions, and check with the system administrator for available port numbers and system names.

Message: `receive_conn_request: Error during read_frame!
 read_frame: Error during fio_read_new_frame!`

Description: An error processing result was received from the *fio_read_new_frame* routine. This is a low-level routine used to read from a socket.

Action: Check the error log. Likely the socket connection was terminated in some manner, potentially by the remote node.

Message: `receive_conn_request: Error during
 check_conn_request!`

Description: An error processing result was received from the *check_conn_request* routine.

Action: Check the error log. An adjacent log message will provide additional information.

Message: `check_conn_request: Error frame type not crf!`

Description: A routine was called to process a Connection Request Frame, but the frame type received was not the correct type.

Action: Contact the sender of the frame, that is, the site attempting to connect. The CD-1.1 protocol is not being adhered to, or the message is malformed.

Message: `check_conn_request: CRF not verified as authentic`

Description: A Connection Request Frame failed authentication.

Action: Verify that the public key used to verify the frame corresponds with the private key used by the sender to sign the frame.

Message: `check_conn_request: Error cdo_unpack_frame!`

Description: An error processing result was received from the *cdo_unpack* routine. This is a low-level routine used to disassemble the Connection Request Frame into a form suitable for the application process.

Action: Contact the sender of the frame, that is, the site attempting to connect. The CD-1.1 protocol is not being adhered to, or the message is malformed.

Message: `receive_conn_request: Error during getpeername!
errno <errno>`

Description: A failure was encountered in the system routine *getpeername*; *<errno>* provides system error number.

Action: Check connecting site to verify the connecting software is not terminating (this is the most likely cause of this error). Potentially, system configuration is problematic. Check par file definitions, and check with the system administrator for the availability of system resources (sockets).

Message: `receive_conn_request: Error check_db_addr station
<sta-name>`

Description: An error processing result was received from the *check_db_addr* routine; using the station name *<sta-name>*.

Action: Check the error log. An adjacent log message will provide additional information.

▼ Troubleshooting

Message: `check_db_addr: Error <errno> gdi_get_ArrayStructs <err-string>!`

Description: A processing error was returned from `gdi_get_ArrayStructs`. The returned error number is `<errno>`, which corresponds to the error message `<err-string>`.

Action: Check that the connecting site is defined in the **alphasite** table of the DBMS with the correct definitions. Check with the database administrator to verify the meaning of error number and error string.

Message: `check_db_addr: Error station <sta-name> address <ip-addr> not found in <tbl-name>`

Description: The name of the station and IP address (as provided by the operating system) does not match definitions in the DBMS table `<tbl-name>`. It is likely a data definition problem in the database.

Action: Check that the connecting site is defined in the **alphasite** table of the DBMS with the correct definitions (especially the IP address).

Message: `receive_conn_request: Error check_station_active station <sta-name>`

Description: An error processing result was received from the `check_station_active` routine.

Action: Check the error log. An adjacent log message will provide additional information.

Message: `check_station_active: Error station is null`

Description: A routine was called with a string containing a connection requestor's name that was undefined.

Action: The error should not occur. Most likely a coding error has been introduced.

Message: `check_station_active: Error invalid station length
<sta-name>`

Description: A routine was called with a string containing a connection requestor's name that was of an unacceptable size.

Action: The error should not occur. Most likely a coding error has been introduced.

Message: `check_station_active: Error <errno> FSOpenFrameStore
<store-name>`

Description: A call to open a Frame Store <store-name> failed via the *FSOpenFrameStore* routine. The *libfs* error number returned was <errno>.

Action: Check par file definition of the *framestore-file* parameter for the correct identification of the Frame Store par file.

Message: `check_station_active: Error <errno> FSOpenFrameSet
<set-name>`

Description: A call to open a frame set <store-name> failed via the *FSOpenFrameSet* routine. The *libfs* error number returned was <errno>.

Action: Check that the *Connection Manager's* par file definition of the current-location parameter is correctly defined. Check the Frame Store configuration par file for a controller frame set corresponding to the connection being attempted. There should be a frame set named *CTLR: IDC-<connector>*.

▼ Troubleshooting

Message: `check_station_active: Error fl_get_latest_logtime`

Description: An error processing result was received from the *fl_get_latest_logtime* routine. A request was made to get the time of the last entry in a frame log and that request failed.

Action: Check the error log. An adjacent log message will provide additional information. This is a return from a low-level library routine and will happen if the log file was inadvertently deleted.

Message: `check_station_active: Error station <sta-name>
active!
check_station_active: Current time=<time-val>
latest_time=<time -val>
check_station_active: Timeout=<sec.s>`

Description: These three log strings are produced as part of a single error condition. A request was received to establish a connection for a site that is already connected.

Action: If the station disconnected and immediately attempted a reconnect, this message will be produced. After a back-off number of seconds has elapsed (<sec.s>), a reconnect may be attempted without suffering this message. If the message appears repeatedly, verify that an existing instance of the requested connection does not exist. If the connection is not active, contact the connecting site and ask them to increase the amount of time between successive connections after a break in an existing communication.

Message: `process_connection_requests: Error in
set_server_connection!`

Description: An error processing result was received from the *set_server_connection* routine.

Action: Check the error log. An adjacent log message will provide additional information.

Message: `set_server_connection: Error`
 `get_array_of_servers_from_db!`
 `get_array_of_servers_from_db: Error <errno>`
 `gdi_get_ArrayStructs <err-string>!`

Description: A processing error was returned from *gdi_get_ArrayStructs*. The returned error number is *<errno>*, which corresponds to the error message *<err-string>*.

Action: Check that the *dlid* identified in the **alphasite** table is defined in the **dlman** table of the DBMS with correct definitions. Check with the database administrator to verify the meaning of error number and error string.

Message: `set_server_connection: Error in lock_station!`

Description: An error processing result was received from the *lock_station* routine.

Action: Check the error log. An adjacent log message will provide additional information.

Message: `lock_station: Error <errno> gdi_get_ArrayStructs`
 `<err-string>!`

Description: A processing error was returned from *gdi_get_ArrayStructs*. The returned error number is *<errno>*, which corresponds to the error message *<err-string>*.

Action: Check that the **alphasite** table of the DBMS is correctly defined with definitions for the expected connection sites. Check with the database administrator to verify the meaning of error number and error string.

▼ Troubleshooting

Message: lock_station: Error over maximum stations!

Description: A data structure size has overflowed, that is, there are more entries being retrieved from the **alphasite** table by the DBMS than the calling routine allocated space to hold.

Action: Contact the software maintenance organization to have the code updated.

Message: lock_station: Error query alphasite zero rows!

Description: A request to get DBMS entries of expected connecting sites resulted in no sites being found.

Action: Verify the **alphasite** table in the database is populated. Also, verify the correct database instance is being accessed. See the *database* parameter definition in the *par* file.

Message: lock_station: station <sta-name> dlid <dl-id> active!

Description: These two log strings are produced as part of a single error condition. The site <sta-name> is/was connected to the host identified by the **dlman** table entry <dl-id>. Another connection request was received from <sta-name>, but the "back-off" number of seconds that would allow the site to reconnect has not elapsed. Time when the connection was attempted is <time-now>. Time when the system last thought the site was connected is <time-last>.

Action: Verify that existing instances of the *Connection Manager* and/or *Connection Manager Server* are not hung-up on their processing. If processing appears correct at the data center, contact the connecting site and ask them to increase the amount of time between successive connections after a break in an existing communication.

```
Message:  lock_station: now time <time-now/> station time
          <time-last>
          update station: Error <errno> gdi execute <err-string>!
```

Description: A processing error was returned from *gdi_execute*. The returned error number is *<errno>*, which corresponds to the error message *<err-string>*. The failure occurred when an attempt was made at updating the **alphasite** table.

Action: Check that the **alphasite** table in DBMS is correctly defined. Check with the database administrator to verify the meaning of error number and error string.

Message: lock station: Error over maximum stations!

Description: A data structure size has overflowed, that is, there are more entries being retrieved from the **alphasite** table by DBMS than the calling routine allocated space to hold.

Action: Contact the software maintenance organization to have the code updated.

Message: lock_station: Set dlid = 0 for sta <sta-name> addr
<ip-addr>!

Description: A failure occurred in attempting to lock the site <sta-name> with an IP address <ip-addr>. The sites are locked while their connection request is being processed to protect the system from multiple connections to a site rapidly issuing connection requests. The message is logged when the software is unable to execute the locking operation.

Action: Contact the database administrator to investigate why this failure has occurred.

▼ Troubleshooting

Message: `set_server_connection: Error in getpeername!
 errno <errno>`

Description: A failure was encountered in the system routine *getpeername*; *errno* provides the system error number.

Action: Check the execution of *Connection Manager Server* (check log file of that process) to verify the process is not terminating, which is the most likely cause of this error. Potentially, the system configuration is problematic. Check par file definitions and check with the system administrator for the availability of system resources, that is, sockets.

Message: `set_server_connection: Error update_station
 <sta-name>!`

Description: An error processing result was received from the *update_station* routine.

Action: Check the error log. An adjacent log message will provide additional information.

Message: `set_server_connection: Unable to find a server!`

Description: From the **alphasite** table, processing will obtain a list of hosts to service a connection request. Each host will be contacted (a *Connection Manager Server* on each host) requesting service for the connection. If none of the hosts respond that they will accept the connection, this message will be produced.

Action: Verify that the *Connection Manager Servers* are correctly configured (par files are correct, port numbers are properly assigned, and so on). Verify that the Internet daemons on the computers hosting the *Connection Manager Servers* are configured to start the process. Verify that the *run_<inst>_connsvr* script in the *.../bin* directory is correctly defined to start the *Connection Manager Server*. Verify that the **alphasite** and **dlman** tables are correctly defined.

Message: `process_connection_requests: Error in
send_conn_response!`

Description: An error processing result was received from the *set_server_connection* routine.

Action: Check the error log. An adjacent log message will provide additional information.

Message: `send_conn_response: Error in create_conn_response!`

Description: An error processing result was received from the *set_server_connection* routine.

Action: Check the error log. An adjacent log message will provide additional information.

Message: `create_conn_response: Error in cdo_build_crs_frame
error <errno>`

Description: An error processing result was received from the *cdo_build_crs_frame* routine. This is a low-level routine used to build a Connection Response Frame.

Action: The error should not occur unless an error has been introduced to the software. Contact the software maintenance organization.

Message: `send_conn_response: Error during write_frame!`

Description: An error processing result was received from the *write_frame* routine.

Action: Check the error log. An adjacent log message will provide additional information.

▼ Troubleshooting

Message: `write_frame: Error writing frame to socket!`

Description: An error processing result was received from the (*libfio*) *safe_write* routine. This is a low-level routine used to write to a socket.

Action: Check the error log. An adjacent log message will provide additional information. Likely the socket connection was terminated in some manner, potentially by the remote node.

Message: `listen_connection: Error during bind_to_port!`

Description: An error processing result was received from the *bind_to_port* routine.

Action: Check the error log. An adjacent log message will provide additional information.

Message: `connect_to_port: Invalid port number!`

Description: Port number received by routine was not in the acceptable range of values.

Action: Check par files of *Connection Manager* and *Connection Manager Server*.

Message: `connect_to_port: Invalid address name!`

Description: A null name string was received by the routine.

Action: Check par files of *Connection Manager* and *Connection Manager Server* for the correct identification of host machines.

Message: `connect_to_port: Error create socket errno = <errno>`

Description: A call to the system routine *socket* returned an error.

Action: Determine meaning of <errno>, the return value. May require the assistance of the system administrator to correct.

Message: `connect_to_port: Error get_ip_address!`

Description: An error processing result was received from the *get_id_address* routine.

Action: Check the error log. An adjacent log message will provide additional information.

Message: `connect_to_port: Error connect <sta-name>
port <port-num> <err-string>`

Description: A call to the system routine *connect* returned an error.

Action: Refer to *<err-string>* in the message, and determine the source of conflict.

Message: `check_port: Error connect_to_port!`

Description: An error processing result was received from the *connect_to_port* routine.

Action: Check the error log. An adjacent log message will provide additional information.

Message: `get_port_from_server: Error write port request!`

Description: An error processing result was received from the (*libfio*) *safe_write* routine. This is a low-level routine used to write to a socket.

Action: Check the error log. An adjacent log message will provide additional information. Likely the socket connection was terminated in some manner, potentially by remote node.

Message: `get_port_from_server: Error during select <errno>`

Description: A call to the system routine *select* returned an error.

Action: Determine the meaning of *<errno>* return value. May require assistance of the system administrator to correct.

▼ Troubleshooting

Message: `read_port_request: Error during select!`

Description: A call to the system routine *select* returned an error.

Action: Check the error log. An adjacent log message will provide additional information.

Message: `read_port_request: Error during safe_read!`

Description: An error processing result was received from the (*libfio*) *safe_read* routine. This is a low-level routine used to read to a socket.

Action: Check the error log. An adjacent log message will provide additional information. Likely the socket connection was terminated in some manner, potentially by remote node.

Message: `read_port_request: Invalid port request message!`

Description: Software was expecting a port request message, but received a message of a different type (potentially a null message).

Action: Verify that a message was not left pending by a previous execution of the process that was terminated unnaturally. If re-execution in a "clean" environment does not eliminate the problem, a software error may have been introduced.

Message: `send_port_response: Error write port response!`

Description: An error processing result was received from the (*libfio*) *safe_write* routine. This is a low-level routine used to write to a socket.

Action: Check the error log. An adjacent log message will provide additional information. Likely the socket connection was terminated in some manner, potentially by remote node.

Message: `initialize_database: Error <errno> gdi_open
<err-string>!`

Description: Failure occurred while opening the database account due to an invalid connection request.

Action: Check par file for the correct specification of the database account/password and instance. Also, check the definition of the environment variables *TWO_TASK* and *GDI_HOME*.

Message: `terminate_database: Error <errno> gdi_commit
<err-string>!`

Description: An error was received from the routine *gdi_commit*.

Action: `<err-string>` identifies a problem was encountered. See the database administrator.

Message: `terminate_database: Error <errno> gdi_close
<err-string>!`

Description: An error was received from the routine *gdi_close*.

Action: `<err-string>` identifies a problem was encountered. See the database administrator.

Connection Manager Server

The *Connection Manager* and *Connection Manager Server* share source code. As a consequence, some of the errors are common between the two processes. Refer to *Connection Manager* errors message as well as those below for the full list of possible *Connection Manager Server* fatal errors.

For messages that begin with either a single word or words connected by an underbar (`_`) and followed immediately by a colon (`:`), the leading word(s) is the name of the function that produced the message.

▼ Troubleshooting

Message: `check_option_request: Error frame type not orf!`

Description: A routine was called to process an Option Response Frame, but frame provided was the wrong type.

Action: Verify a message was not left pending by a previous execution of process that was terminated unnaturally. If re-execution in a “clean” environment does not eliminate the problem, then a software error may have been introduced.

Message: `check_option_request: ORF not verified as authentic`

Description: Signature verification of the Option Request Frame failed.

Action: Check that the correct public key for the connecting site has been installed and that the *index.txt* file for the public key directory has been updated correctly. Unlikely this error would occur because an authenticated Connection Request Frame from the requesting site would have proceeded the Option Request Frame and would have been validated.

Message: `check_option_request: Error cdo_unpack_frame!`

Description: An error was received from the *cdo_unpack_frame* routine. This would occur if an incorrectly formatted frame has been received.

Action: Contact the requesting site to have them verify frame construction.

Message: `check_option_request: ORF body count is zero!`

Description: An incomplete Option Request Frame was received for establishing a connection.

Action: Contact the requesting site to have them verify frame construction.

Message: `check_option_request: ORF option size less than one!`

Description: An incomplete Option Request Frame was received for establishing a connection.

Action: Contact the requesting site to have them verify frame construction.

Message: `check_option_request: Invalid station in orf`

Description: A site identified in the Option Request Frame is different than the one received by the function in the calling argument.

Action: Check the contents of the *port_par_file*. Contact the requesting site to verify frame construction. Processing uses the length of the string in verification. If extraneous characters are supplied it could cause this error.

Message: `Must have at least two arguments`

Description: An incorrect invocation of the process. The needed configuration arguments are not provided on the command line.

Action: Use the correct command line for startup.

Message: `ConnMgr_server: Error initializing log file!`

Description: Unable to create or write to the log file.

Action: Verify the log directory specified in the configuration exists and is write-able by the user.

▼ Troubleshooting

Message: `initialize: Error initialize_parameters!`

Description: Unable to read or parse the configuration parameters from the par file.

Action: Check the format and syntax of the par file variable definitions. There may be another adjacent log message that specifies which parameter was problematic.

Message: `initialize: Error initialize_signal_handler!`

Description: An error was encountered in establishing a system signal handler.

Action: See the system administrator. There may be another adjacent log message that will provide additional information.

Message: `initialize: Error initialize_authentication!`

Description: An error was encountered in the initialization of authentication software.

Action: Check the pathname to the authentication directory hierarchy provided in the configuration file. There may be another adjacent log message that will provide additional information.

Message: `initialize_parameters: Error reading
well_known_port parameter!`

Description: A required parameter was not defined or was incorrectly defined.

Action: Verify the definition of `well_known_port` in the par file.

Message: `initialize_parameters: port_par_file not a
parameter!`

Description: A required parameter was not defined or was incorrectly defined.

Action: Verify the definition of `port_par_file` in the par file.

Message: `initialize_parameters: exch_con_path not a parameter!`

Description: A required parameter was not defined or was incorrectly defined.

Action: Verify the definition of `exch_con_path` in the par file.

Message: `initialize_parameters: exch_con_param_file not a parameter`

Description: A required parameter was not defined or was incorrectly defined.

Action: Verify the definition of `exch_con_param_file` in par file.

Message: `initialize_parameters: Error inet parameter must be 0 or 1`

Description: A configuration parameter was improperly defined.

Action: Edit the par file to provide the acceptable parameter value (0 or 1).

Message: `SIGPIPE signal received`

Description: A terminating signal was received from the operating system.

Action: None. The *Connection Manager* will be restarted by the Internet daemon at the next connection request. If the problem reoccurs, contact system administrator.

Message: `SIGTERM signal received`

Description: A terminating signal received from the operating system.

Action: None. The *Connection Manager* will be restarted by the Internet daemon at the next connection request. If the problem reoccurs, contact the system administrator.

▼ Troubleshooting

Message: SIGALRM signal received

Description: A terminating signal was received from the operating system.

Action: None. The *Connection Manager* will be restarted by the Internet daemon at the next connection request. If the problem reoccurs, contact the system administrator.

Message: Error signal <sig-num> received

Description: A terminating signal was received from the operating system.

Action: None. The *Connection Manager* will be restarted by the Internet daemon at the next connection request. If the problem reoccurs, contact the system administrator.

Message: initialize_authentication: Error
process_logging_filedes!
process_logging_filedes: Error fdopen to set AS
logging

Description: An invalid file descriptor was received for logging file access.

Action: Check definitions in the configuration files. The error should not occur and may indicate a coding error.

Message: process_port_request: Error during
accept_connection!

Description: An error processing result was received from the *accept_connection* routine.

Action: Check the error log. An adjacent log message will provide additional information.

Message: `process_port_request: Error in read_port_request!`

Description: An error processing result was received from the *read_port_request* routine.

Action: Check the error log. An adjacent log message will provide additional information.

Message: `process_port_request: Error in get_port_number!`

Description: An error processing result was received from the *get_port_number* routine.

Action: Check the error log. An adjacent log message will provide additional information.

Message: `get_port_number: Error invalid port number!`

Description: A port number outside the legal range of values (*val* < 0 or > 99999) was read from the port par file.

Action: Update the par file to contain the legal port values. The port number used should be coordinated with the system administrator.

Message: `process_port_request: Error during listen_connection!`

Description: An error processing result was received from the *listen_connection* routine.

Action: Check the error log. An adjacent log message will provide additional information.

▼ Troubleshooting

Message: `process_port_request: Error in send_port_response!`

Description: An error processing result was received from the *send_port_response* routine.

Action: Check the error log. An adjacent log message will provide additional information.

Message: `process_option_request: Error in receive_option_request!`

Description: An error processing result was received from the *receive_option_request* routine.

Action: Check the error log. An adjacent log message will provide additional information.

Message: `receive_option_request: Error during accept_on_socket!`

Description: An error processing result was received from the *accept_on_socket* routine.

Action: Check the error log. An adjacent log message will provide additional information.

Message: `receive_option_request: Error during select <errno>`

Description: A call to the system routine *select* was produced, and an error was returned.

Action: Determine the meaning of *<errno>*, the return value. May require the assistance of the system administrator to correct.

Message: `receive_option_request: Error during read_frame!`

Description: An error processing result was received from the *read_frame* routine.

Action: Check the error log. An adjacent log message will provide additional information.

Message: `receive_option_request: Error in check_option_request!`

Description: An error processing result was received from the *check_option_request* routine.

Action: Check the error log. An adjacent log message will provide additional information.

Message: `process_option_request: Error in send_conn_response!`

Description: An error processing result was received from the *send_option_response* routine. The name of the routine is incorrect in the message. However, the correct routine will have been called and will have provided additional logging information.

Action: Check the error log. An adjacent log message will provide additional information.

Message: `send_option_response: Error in create_option_response!`

Description: An error processing result was received from the *create_option_response* routine.

Action: Check the error log. An adjacent log message will provide additional information.

▼ Troubleshooting

Message: `create_option_response: Error in
cdo_build_oaf_frame!`

Description: An error was received from the *cdo_build_oaf_frame* function.

Action: A software error has been introduced. Verify recent changes.

Message: `send_option_response: Error during write_frame!`

Description: An error processing result was received from the *write_frame* routine.

Action: Check the error log. An adjacent log message will provide additional information.

Message: `process_option_request: Error in
start_frame_exchange!`

Description: An error processing result was received from the *start_frame_exchange* routine.

Action: Check the error log. An adjacent log message will provide additional information.

Message: `start_frame_exchange: Error dup file descriptor!`

Description: An error was received from the system function duplicate file descriptor.

Action: Likely the limit of open file descriptors has been reached. Limit value can be determined by entering the UNIX command `limit`.

Message: `start_frame_exchange: Error reading par file!`

Description: Message not exactly correct. The code could not set the '*par = <some path>*' string from the variable *exch_con_param_file*, which should have been set from the *par* file value.

Action: Check the *par* file to verify a valid pathname is provided for the *Exchange Controller's* *par* file string. A software error may have been

introduced. If the code was able to read the *par* variable, it should have been able to set it in the *sprintf* call.

Message: `start_frame_exchange: Error reading file descriptor!`

Description: An error was received when doing a *sprintf* on a file descriptor value.

Action: A software error has been introduced.

Message: `start_frame_exchange: Error reading connection name!`

Description: An error was received when doing *sprintf* on the *req->station_name* variable.

Action: A software error has been introduced. A variable had been verified as valid earlier in processing flow.

Message: `start_frame_exchange: Error during fork!`

Description: An error was received from the system routine *fork*.

Action: Check that the number of active processes limit has not been exceeded.

Message: `start_frame_exchange: Error exec Exchange Controller!`

Description: A error was received from the system routine *execv*.

Action: Check the *par* file for the valid pathname to the *Exchange Controller* process. Also, check that the process has permissions that allow execution by *CDS CD-1.1* user.

▼ Troubleshooting

Connection Originator

For messages that begin with either a single word or words connected by an underbar (_) and followed immediately by a colon (:), the leading word(s) is the name of the function that produced the message.

Message: `process_logging_filedes: 10009 : couldn't associate stream with file descriptor <n> (<err-str>)`

Description: The software could not assign the logging file descriptor variable. The message `<n>` token provides attempted file descriptor value; `<err-str>` provides *errno* result string.

Action: Investigate system resources based on the content of `<err-str>` token.

Message: `check_required_string_par: 10015: missing required parameter <par-str>`

Description: A required par file value is missing.

Action: Provide a definition for the parameter identified in the `<par-str>` token in the par file.

Message: `create_connection: 10002: couldn't CD-connect`

Description: Failed to connect to the data consumer computer.

Action: Check the par file for the correct identification of the data consumer computer and port number. Verify that the remote computer is up.

Message: `create_connection: 10016: couldn't CD-connect`

Description: Failed to connect to data consumer computer.

Action: Check the par file for the correct identification of the data consumer computer and port number. Verify that the remote computer is up.

Message: `co_exchcntl_exec: execv of exchange controller failed`

Description: An error was received from the system routine `execv`.

Action: Check that the par file for the valid pathname to the *Exchange Controller* process. Also, check that the process has permissions that allow execution by the *CDS CD-1.1* user.

Message: `build_ec_args: 10001: can't build args to exec exchange controller`

Description: An error was encountered building the command string used to start the *Exchange Controller*.

Action: Check that the par file contains the valid parameter definitions for the *Exchange Controller* path, and check the *Exchange Controller* par variable list.

Message: `argv_init: 10001: can't allocate memory`

Description: An error was received from the system call `calloc`.

Action: A software error may have been introduced, or the par file parameter providing the par list for *Exchange Controller* is corrupt. Code failed trying to allocate space for a character string pointer.

Message: `argv_add: 10001: can't allocate memory`

Description: An error was received from the system call `strdup`.

Action: A software error may have been introduced, or the par file parameter providing the par list for *Exchange Controller* is corrupt.

▼ Troubleshooting

Message: `argv_incr: 10001: can't allocate memory`

Description: An error was received from the system call *realloc*.

Action: A software error may have been introduced, or the par file parameter providing the par list for the *Exchange Controller* is corrupt.

Message: `co_frames_pack_orf: 10017: size of option request item exceeds <n>`

Description: An error constructing the Option Request Frame was received. Size of field has been exceeded.

Action: Check the par file value for *co-optreq-item* or, if not defined, *station_name*. Length should be less than 1,024.

Message: `co_frames_verify_frame: 10018: couldn't verify frame signature`

Description: Authentication failed on the frame received from the data consumer.

Action: Verify that the correct public key has been installed in the authentication directory hierarchy and the *index.txt* file correctly identifies the public key.

Message: `co_portinfo_get: 10001: can't allocate memory`

Description: An error was received from the system call *calloc*.

Action: A software error may have been introduced, or the par file parameter *co-max-connmgr-port* is corrupt.

Message: `add_one_port_addr: 10012: exceeded <n> IP addresses, increase value of co-max-connmgr-ports`

Description: An error occurred while creating the list of ports used in connection attempts.

Action: Check the definition of *co-max-connmgr-ports* in the par file.

Message: `add_one_port_addr: 10001: can't allocate memory`

Description: An error was received from the system call *malloc*.

Action: A software error may have been introduced, or the par file parameter *co-max-connmgr-port* is corrupt.

Data Parser

Message: `DLParse – must specify dlid (diskloop id)`

Description: The required par file parameter was not found.

Action: Add a definition for *dlid* parameter in the par file.

Message: `DLParse – Must specify database!`

Description: The required par file parameter was not found.

Action: Add a definition for the database parameter in the par file to identify the DBMS user-name/password and instance.

Message: `FATAL – must specify machine`

Description: The required par file parameter was not found.

Action: Add the definition for the *machine* parameter in the par file to identify the host computer for this instance of *Data Parser*.

Message: `wfdisc moves not allowed`

Description: The *allow_wfdisc_move* parameter in the par file was set to true, but is not supported by *Data Parser* process.

Action: Update the *allow_wfdisc_move* parameter in the par file to false/0.

▼ Troubleshooting

Message: Cannot access *ca-cert-path* <*ca-path*>

Description: The par file variable *ca-cert-path* specified a directory path inaccessible to software.

Action: Update the *auth_<dld>.par* file definition and update the directory hierarchy to provide accessibility to the certificate authority or certificate directory.

Message: Missing password for authentication system

Description: The par file variable *auth-pass-phrase* is missing.

Action: Provide the needed definition in the *auth_<dld>.par* file.

Message: Bad *sta/frameset* name: <*name-string*>

Description: The par file parameter *station_fs_string* is incorrectly defined.

Action: Correct the definition of the parameter in the par file. Parameter should be a list of sites and frame set name pairs, with each token separated by space or comma.

Message: Missing *fs_name*.

Description: The par file parameter *framestore_config* is not defined.

Action: Update the par file to provide a definition; parameter should identify full path to the Frame Store configuration par file.

Message: FATAL: DLMan identified by *dld* %d must run on <*host-name*> - exit

Description: *Data Parser* instance has been started on the computer that differs from the name provided in the par file parameter *machine*.

Action: Update the configuration to make the execution instance and machine par file parameter agree.

Message: FSOpenFrameStore (<config-file>) failed, err= <err-nbr>

Description: A request to open the Frame Store identified by <config-file> failed with the error as provided in the *err-nbr*.

Action: Verify the definition of the *framestore_config* parameter in the par file. Verify the directories and files of the Frame Store (identified in par file) exist with correct permissions for the *CDS CD-1.1* processes.

Message: DLParse FATAL – cannot open database

Description: Process was unable to open the database account specified by the *database* parameter from the par file.

Action: Verify that the *database* parameter is correctly defined in the par file. Check the database accounts and configuration.

Message: DLParse FATAL – Cannot add dlman data to db

Description: The process was unable to update the **dlman** table.

Action: Check the configuration of the database. Check for stuck processes that have the table locked.

Message: DLParse FATAL – db error accessing dlman

Description: The process was unable to access the **dlman** table.

Action: Check the configuration of the database. Check for stuck processes that have the table locked.

Message: DLParse FATAL – cannot update dlman status in db

Description: The process was unable to update the **dlman** table.

Action: Check the configuration of the database. Check for stuck processes that have the table locked.

▼ Troubleshooting

Message: No stations to process - exiting

Description: Processing could not find the frame sets for sites provided in the *station_fs_string* par file parameter.

Action: Update the *station_fs_string* parameter/Frame Store configuration par file to identify the desired sites and their frame sets.

Message: DLParse - input_loop returned!

Description: Debugging output.

Action: If this message is produced during normal operations, there will be adjacent messages in the log file providing additional information.

Message: Cannot flush database buffers - exiting now

Description: An error was encountered when writing to the database.

Action: Adjacent logging messages will provide additional information about what write operation was being attempted.

Message: unrecoverable error in find_loop

Description: Process unable to find correct the Disk Loop file for writing data.

Action: Check the **dlfile** table for distinct rows having duplicate time value for station/channel. If no duplicate exists, restart. If a duplicate exists, set time to 0 for one of the rows (some data will be lost).

Message: can't insert at start

Description: The routine *write_data* was unable to write waveforms to the Disk Loops. **WFWR_INVALID** was returned from the *insert_wf* routine.

Action: None. Wait for the *Data Parser* to restart. The problem was either a Data Frame that overlapped preceding Data Frames or an algorithm error with **wfdisc** records in memory. Restarting clears out either problem.

Message: `append_wf failed`

Description: The routine *write_data* was unable to write waveforms to the Disk Loops. `WFWR_INVALID` was returned from the *append_wf* routine.

Action: None, wait for the *Data Parser* to restart. The problem was either a Data Frame that overlapped preceding Data Frames or an algorithm error with **wfdisc** records in memory. Restarting clears out either problem.

Message: `prepend_wf failed`

Description: The routine *write_data* was unable to write waveforms to Disk Loops. `WFWR_INVALID` was returned from the *prepend_wf* routine.

Action: None, wait for the *Data Parser* to restart. The problem was either a Data Frame that overlapped preceding Data Frames or an algorithm error with **wfdisc** records in memory. Restarting clears out either problem.

Message: `insert_wf failed`

Description: The routine *write_data* was unable to write waveforms to the Disk Loops. `WFWR_INVALID` was returned from the *insert_wf* routine.

Action: None, wait for the *Data Parser* to restart. The problem was either a Data Frame that overlapped preceding Data Frames or an algorithm error with **wfdisc** records in memory. Restarting clears out either problem.

▼ Troubleshooting

Message: `append_wf failed`

Description: The routine *write_data* was unable to write waveforms to the Disk Loops. `WFWR_INVALID` was returned from the *append_wf* routine.

Action: None, wait for the *Data Parser* to restart. The problem was either a Data Frame that overlapped preceding Data Frames or an algorithm error with `wfdisc` records in memory. Restarting clears out either problem.

Data Center Manager

In messages that begin with either a single word or words connected by an underbar (_) and followed immediately by a colon (:), the leading word(s) is the name of the function that produced the message.

Message: `start_threads: 15: can't create terminator thread (<err-nbr>)`

Description: An error was received from the system routine *pthread_create*. The `<err-nbr>` token provides an error number from the routine.

Action: A software error may have been introduced. Otherwise, check the system resource usage.

Message: `start_threads: 15: can't create reaper thread (<err-nbr>)`

Description: An error was received from the system routine *pthread_create*. The `<err-nbr>` token provides an error number from the routine.

Action: A software error may have been introduced. Otherwise, check the system resource usage.

Message: `start_threads: 15: can't create sleeper thread
(<err-nbr>)`

Description: An error was received from the system routine *pthread_create*. The `<err-nbr>` token provides an error number from the routine.

Action: A software error may have been introduced. Otherwise, check the system resource usage.

Message: `start_threads: 15: can't create fdwatcher thread
(<err-nbr>)`

Description: An error was received from the system routine *pthread_create*. The `<err-nbr>` token provides an error number from the routine.

Action: A software error may have been introduced. Otherwise, check the system resource usage.

Message: `init_connreq_sockfd: 13: can't create socket`

Description: An error was received from the system routine *socket*.

Action: Check that user resource limits have not been exceeded, particularly the number of file descriptors.

Message: `init_connreq_sockfd: 13: can't bind address
<ip-addr>, port <port-nbr> to command and
control socket <socket-fd>`

Description: An error was received from the system routine *bind*. The process failed binding to the socket that would have been used to receive communications from the children processes.

Action: Check that there is no conflict with the use of port numbers or file descriptor system resources.

▼ Troubleshooting

Message: `init_connreq_sockfd: 13: can't set backlog
<backlog-nbr> for c-and-c port <socket_fd>`

Description: An error was received from the system routine *listen*. The *<backlog-nbr>* token set by par file parameter *fm-c-and-c-backlog* or default of *DEFAULT_C_AND_C_BACKLOG* is incorrect.

Action: A software error may have been introduced. Otherwise, check the system resource usage.

Message: `handle_runjob_event: 03: can't fork`

Description: An error was received from the system routine *fork* when trying to start a process to be managed.

Action: Check that user resource limits have not been exceeded and a software error has not been introduced.

Message: `accept_new_c_and_c_connection: 12: can't set new
connection sockfd <socket-fd> to non-blocking.`

Description: An error was received from the system routine *fcntl*.

Action: Check that user resource limits have not been exceeded and a software error has not been introduced.

Message: `read_and_log_fd_input: 12: error reading file
(errno <err-nbr>)`

Description: An error was received from the system routine *read* when reading the input pipe from the child process. The error is a value less than zero for the number of bytes read. The *<err-nbr>* token contains the system *errno* value from the call.

Action: Investigate the error based on the value of the *<err-nbr>* token.

Message: `evq_init_eventqueue: 15 : error initializing event queue mutex`

Description: An error was received from the system routine *pthread_mutex_init*.

Action: Check that user resource limits have not been exceeded and a software error has not been introduced.

Message: `evq_init_eventqueue: 15 : error initializing event queue condition variable`

Description: An error was received from the system routine *pthread_cond_init*.

Action: Check that user resource limits have not been exceeded and a software error has not been introduced.

Message: `evq_dequeue_event: 15 : error locking eventqueue mutex`

Description: An error was received from the system routine *pthread_mutex_lock*.

Action: Check that a software error has not been introduced.

Message: `evq_dequeue_event: 15 : error waiting on eventqueue condition variable`

Description: An error was received from the system routine *pthread_cond_wait*.

Action: Check that a software error has not been introduced.

Message: `evq_dequeue_event: 15 : error unlocking eventqueue mutex`

Description: An error was received from the system routine *pthread_mutex_unlock*.

Action: Check that a software error has not been introduced.

▼ Troubleshooting

Message: evq_enqueue_event: 15 : error locking eventqueue mutex

Description: An error was received from the system routine *pthread_mutex_lock*.

Action: Check that a software error has not been introduced.

Message: evq_enqueue_event: error signalling eventqueue condition variable

Description: An error was received from the system routine *pthread_cond_signal*.

Action: Check that a software error has not been introduced.

Message: evq_enqueue_event: 15 : error unlocking eventqueue mutex

Description: An error was received from the system routine *pthread_mutex_unlock*.

Action: Check that a software error has not been introduced.

Message: evq_push_event: 15 : error locking eventqueue mutex

Description: An error was received from the system routine *pthread_mutex_lock*.

Action: Check that a software error has not been introduced.

Message: evq_push_event: 15 : error signalling eventqueue condition variable

Description: An error was received from the system routine *pthread_cond_signal*.

Action: Check that a software error has not been introduced.

Message: `evq_push_event: 15 : error unlocking eventqueue mutex`

Description: An error was received from the system routine *pthread_mutex_unlock*.

Action: Check that a software error has not been introduced.

Message: `fdw_fdwatcher_main_loop: 16 : select error (errno == <err-nbr>)`

Description: An error was received from the system routine *select*. The *<err-nbr>* token provides the system error number from the error return.

Action: Investigate the error based on the value of the *<err-nbr>* token.

Message: `add_new_watched_fds: 15 : can't lock mutex`

Description: An error was received from the system routine *pthread_mutex_lock*.

Action: Check that a software error has not been introduced.

Message: `get_unused_job_ndx: 05 : number of jobs exceeds <nbr>, exiting`

Description: The request for the number of managed jobs exceeds the configured number (given by *<nbr>* token).

Action: Check that a software error has not been introduced. Increase the number of jobs supported by *ForeMan*, which is set by the par file parameter *fm-max-number-jobs*.

▼ Troubleshooting

Message: get_job_template: 06 : undefined job template id
 <tmpl-string>

Description: An inconsistent request was received to manage the job with the par file template <tmpl-string>.

Action: Examine the par file for consistent definitions between the list of job templates *fm-job-template-ids* and the defined templates.

Message: build_exec_vector: 01 : i: <args-attempted>,
 total_num_args: <args-needed>

Description: An inconsistency was encountered in starting a managed process. The number of command line arguments differs.

Action: Check that a software error has not been introduced.

Message: parse_event_from_string: 07 : missing event string
 terminator (<term-str>) in string <tmpl-str>

Description: An error occurred while parsing an event string. An incorrectly formatted job control string was received. The expected terminator string <term-str> was not found in the string <tmpl-str>.

Action: Verify that the correct formatting of the job template string in the par file/job control strings was produced by the managed processes (for consumption by *ForeMan*).

Message: rpr_reaper_main_loop: 01 : wait() failed, error
 <err-nbr>

Description: An error was received from the system routine *wait*. The <err-nbr> token provides the system error number from the error return.

Action: Investigate the error based on the value of the <err-nbr> token.

Message: `trm_terminator_main_loop: can't enqueue haltreq event for graceful exit!`

Description: An error was received during the termination processing from either the *evt_make_haltreq_event* or *evq_push_event* routine.

Action: Check the log file for adjacent log messages that provide additional information.

Frame Exchange

Unlike messages from other components, the first word in a *Frame Exchange* message immediately followed by a colon (:) does not provide the name of a function, but is intended to denote the functional component within the program.

Message: `fex: Fatal: par variable SOCKFD, undefined`

Description: The required command line argument `SOCKFD` was not supplied in the process invocation.

Action: *FrameEx* is invoked by *Exchange Controller*. Check that a software error has not been introduced to the *Exchange Controller*.

Message: `fex: Fatal: par variable PIPE_TO_CTLR, undefined`

Description: The required command line argument `PIPE_TO_CTLR` was not supplied in the process invocation.

Action: *FrameEx* is invoked by *Exchange Controller*. Check that a software error has not been introduced to the *Exchange Controller*.

▼ Troubleshooting

Message:	fex: Fatal: par variable PIPE_FROM_CTLR, undefined
Description:	The required command line argument PIPE_FROM_CTLR was not supplied in process invocation
Action:	<i>FrameEx</i> is invoked by <i>Exchange Controller</i> . Check that a software error has not been introduced to the <i>Exchange Controller</i> .

Message:	fex: Not enough memory to start (<nbr> framesets)
Description:	An error was received from the <i>malloc</i> system call allocating memory for the gap lists and queues. The <nbr> presents the number of frame sets <i>FrameEx</i> is attempting to support.
Action:	If <nbr> is correct, investigate the number of active processes on the host, and/or contact the system administrator about increasing the swap space. If <nbr> is not correct, investigate the par file and <i>Frame Store</i> par file to verify the definitions of the needed frame sets.

Message:	fex: Frame exchange initialization failed
Description:	An error was received from the routine <i>init_frame_exchange</i> .
Action:	Check the error log. An adjacent log message will provide additional information.

Message:	fex: Cannot init mutexes
Description:	An error was received from the system routine <i>pthread_mutex_init</i> .
Action:	Check that a software error has not been introduced.

Message: `fex: Local gap <set-nbr> init failed`

Description: The initialization of the frame gap list failed. The gap list identifies gaps in sequence numbers for frames received in a frame set. The `<set-nbr>` token identifies an ordinal number within *FrameEx* for a frame set.

Action: Check the error log. An adjacent entry will provide additional information. Check that a software error has not been introduced.

Message: `fex: Peer gap <set-nbr> init failed`

Description: The initialization of the frame gap list failed. *FrameEx* exchanges gap lists with the protocol peer. Gap lists identify gaps in sequence numbers for the received frames in a frame set. When the peer does not provide a gap list, or provides one that is corrupt, this message is displayed. The `<set-nbr>` token identifies an ordinal number within *Frame Exchange* for a frame set.

Action: Check that a software error has not been introduced. Contact the peer node to verify that a valid *Frame Exchange* or *Frame Exchange-like* process is in use.

Message: `fex: Sender creation failed: <err-nbr>`

Description: An error was received from the system routine *pthread_create*. The `<err-nbr>` token provides the error number from the routine.

Action: A software error may have been introduced. Otherwise, check the system resource usage.

Message: `fex: Heartbeat creation failed: <err-nbr>`

Description: An error was received from the system routine *pthread_create*. The `<err-nbr>` token provides the error number from the routine.

Action: A software error may have been introduced. Otherwise, check the system resource usage.

▼ Troubleshooting

Message: `fex: Message sender creation failed: <err-nbr>`

Description: An error was received from the system routine *pthread_create*. The `<err-nbr>` token provides the error number from the routine.

Action: A software error may have been introduced. Otherwise, check the system resource usage.

Message: `fex: Time counter creation failed: <err-nbr>`

Description: An error was received from the system routine *pthread_create*. The `<err-nbr>` token provides the error number from the routine.

Action: A software error may have been introduced. Otherwise, check the system resource usage.

Message: `fex: Q <set-nbr> mutex init failed`

Description: An error was received from the system routine *pthread_mutex_init*. The `<set-nbr>` token identifies an ordinal number within *Frame Exchange* for the frame set.

Action: Check the user resource limits have not been exceeded and a software error has not been introduced.

Message: `fex: Memory exhaustion`

Description: An error was received from the system routine *malloc*.

Action: Check that a software error has not been introduced. Also check the system configuration for the adequate swap space definition.

Message: `fex: Gap <set-nbr> mutex init failed`

Description: An error was received from the system routine *pthread_mutex_init*. The `<set-nbr>` token identifies an ordinal number within *Frame Exchange* for a frame set.

Action: Check that the user resource limits have not been exceeded and a software error has not been introduced.

Message: `fex_gap: No space to increase gap table`

Description: Failure returned from the routine *double_gaps*. Gap list has increased beyond its initial sizing; the software attempted to double its size.

Action: Investigate why so many sequence number gaps exist. Is the frame creator assigning numbers sequentially? Is the communications link bad? Check the error log for adjacent entries providing additional information. Memory can be increased with additional swap space. However, the source of gaps must be determined.

Message: `fex_gap: No space to increase gap table`

Description: Failure return received from the routine *double_gaps*. Gap list has increased beyond its initial sizing; the software attempted to double its size.

Action: Investigate why so many sequence number gaps exist. Is the frame creator assigning numbers sequentially? Is the communications link bad? Check the error log for adjacent entries providing additional information. Memory can be increased with additional swap space. However, the source of the gaps must be determined.

▼ Troubleshooting

Exchange Controller

For messages that begin with either a single word or words connected by an underbar (_) and followed immediately by a colon (:), the leading word(s) is the name of the function that produced the message.

Message: `init_controller_exec: 12001: Par variable
 frame-exchange-path, undefined`

Description: The required par file parameter is missing.

Action: Provide a needed definition in the par file.

Message: `init_controller_exec: 12002: Par variable
 frame-exchange-par-file, undefined`

Description: The required par file parameter is missing.

Action: Provide a needed definition in the par file.

Message: `init_controller_exec: 12003 : Unable to put par file
 in frame exchange command line`

Description: An error was received from the call to system routine *sprintf* in the building command.

Action: Check the definition of *frame-exchange-par-file* in the par file.

Message: `init_controller_exec: 12004 : Par variable SOCKFD,
 undefined`

Description: The required command line parameter is missing.

Action: *Exchange Controller* is invoked by either the *Connection Manager Server* or *Connection Originator* and is based on whether an inbound or outbound connection is to be supported, respectively. Check that a software error has not been introduced to either one of those programs.

Message: `init_controller_exec: 12005 : Unable to put sock
 descr in frame exchange command line`

Description: An error was received from the call to the system routine *sprintf* in building the command.

Action: Check that a software error has not been introduced.

Message: `init_controller_exec: 12006 : pipe creation failed`

Description: An error was received from the call to the system routine *pipe* to create a communication pipe to the *Frame Exchange*.

Action: Check that user resource limits have not been exceeded and that a software error has not been introduced.

Message: `init_controller_exec: 12007 : pipe creation failed`

Description: An error was received from the call to the system routine *pipe* to create a communication pipe to the *Frame Exchange*.

Action: Check that user resource limits have not been exceeded and a software error has not been introduced.

Message: `init_controller_exec: 12008 : Unable to put pipe
 descr in frame exchange command line`

Description: An error was received from the call to the system routine *sprintf* in building the command.

Action: Check that a software error has not been introduced.

Message: `init_controller_exec: 12009 : Unable to put pipe
 descr in frame exchange command line`

Description: An error was received from the call to the system routine *sprintf* in building the command.

Action: Check that a software error has not been introduced.

▼ Troubleshooting

Message: `init_controller_exec: 12010 : Unable to fork for
Frame Exchange`

Description: An error was received from the call to the system routine *fork* for creating a *Frame Exchange* process.

Action: Check that the user resource limits have not been exceeded and that a software error has not been introduced.

Message: `init_controller_exec: 12011: Unable to exec Frame
Exchange process`

Description: An error was received from the call to the system routine *execv* for creating a *Frame Exchange* process.

Action: Check the par file for valid pathname to the *Frame Exchange* process. Also, check that the process has permissions that allow execution by the *CDS CD-1.1* user.

Message: `controller_exec_sig_catch: 12012 : Broken pipe
signal raised`

Description: A terminating signal was received from the operating system.

Action: None. The *Exchange Controller* will be restarted using the same method by which it was previously invoked. Check the *Frame Exchange* error log for clues as to why the process terminated (the pipe was to *Frame Exchange*).

Message: `controller_exec_sig_catch: 12013 : Terminate signal
received, processing exit`

Description: A terminating signal was received from the operating system.

Action: None. The *Exchange Controller* will be restarted using the same method by which it was previously invoked.

Message: `init_exchange_interface: 12017 : timed out waiting for Frame exchange init`

Description: The *Exchange Controller* expects to receive a message from the *FrameEx* at initialization to signify it is ready for execution. The message was not received within the time period allotted.

Action: Verify that the *Frame Exchange* gets activated. Check the *Frame Exchange* log file to see if abnormalities have occurred. Check the value of the par file parameter `ctlr-msg-limit` that defines how many seconds to wait for the *Frame Exchange* message.

Message: `init_frame_sets: 12021: par variable ctlr-<type>-frame-set-names, undefined, provides unacceptable name list, or produces list exceeding fset capacity`

Description: Definition of par file parameter is invalid or frame set capacity of the *Exchange Controller* is being exceeded by the par file configuration.

Action: Verify the par file definitions/compare the number of frame sets with the *Exchange Controller* constant `MAX_FRAME_SETS`.

Message: `init_frame_handler: 12022 : Failed creation of frm_status_tbl, unable to continue.`

Description: Unable to create in-memory hash table for tracking frame status.

Action: Check that user resource limits have not been exceeded and a software error has not been introduced.

Message: `init_frame_handler: 12023 : par variable ctlr-fs-par-file, undefined`

Description: The required par file parameter was missing. The parameter needed to identify the Frame Store configuration par file is needed.

Action: Update the par file to provide the needed definition.

▼ Troubleshooting

Message: `log_frame_queued: 12024 : Unable to create memory table entry for logging`

Description: An attempt to create the table entry to the log frame status failed. This can happen if the table gets extremely large and the process runs out of memory (failed `malloc`).

Action: The names are removed from the table when they are acknowledged by the receiving peer node. The table size should be small, containing only the outstanding unacknowledged frames (20–40 entries). Determine why the acknowledgements are not being provided.

Message: `resend_queued_frames: 12024 : Unable to create memory table entry for logging`

Description: An attempt to create the table entry to the log frame status failed. This can happen if the table gets extremely large and the process runs out of memory (failed `malloc`).

Action: Frames are removed from the table when they are acknowledged by the receiving peer node. The table size should be small, containing only outstanding unacknowledged frames (20–40 entries). Determine why the acknowledgements are not being provided.

Message: `resend_queued_frames: 12026 : Failure encountered on query for frame status during initialization error string is: <err-str>`

Description: An error was received from the routine to inquire after a frame's status. The `<err-str>` token provides the routine's assessment of the error condition.

Action: Evaluate the meaning of `<err-str>`, and correct any discrepancies.

Message: `init_frame_handler: 12027 : unable to open frame store with par file <par-path>, reported error is: <err-str>.`

Description: The process was unable to open the Frame Store identified by the configuration file `<par-path>`. The `<err-str>` token provides errors reported by the Frame Store open software.

Action: Evaluate the meaning of `<err-str>`, and correct any discrepancies. Verify the location of Frame Store and the permissions of directories and files.

Message: `init_frame_sets: 12028 : unable to open frame set <set-name>, reported error is <err-str>.`

Description: Could not open the frame set given by `<set-name>` token. The `<err-str>` provides the reported error condition.

Action: Evaluate the meaning of `<err-str>`, and correct any discrepancies. Verify the configuration of the Frame Store; does a frame set exist with the proper log file and permissions to allow needed access?

Message: `init_frame_sets: 12029 : Unable to locate the logging frame set, cannot continue`

Description: The frame set for logging the status of sent frames was not defined.

Action: Update the par file to provide the needed parameter; the name is provided in the `ctlr-logging-frame-set-list` parameter.

Message: `init_cmd_frame_params:12047 : undefined par variable, ctlr-cmd frame-set`

Description: The required par file parameter is missing.

Action: Update the par file to provide the needed definition.

▼ Troubleshooting

Message: `init_cmd_frame_params: 12048 : undefined par
 variable ctrlr-cmd-src-id`

Description: The required par file parameter is missing.

Action: Update the par file to provide the needed definition.

Message: `init_cmd_frame_params: 12049 : undefined par
 variable ctrlr-cmd-dest-id`

Description: The required par file parameter is missing.

Action: Update the par file to provide the needed definition.

Message: `init_cmd_frame_params: 12051 : undefined par
 variable ctrlr-cmd-series-id`

Description: The required par file parameter is missing.

Action: Update the par file to provide the needed definition.

Message: `init_exchange_interface: 12065: Failure in
 establishing interface to frame_exchange`

Description: An error occurred attempting to get an initialization coordination message from the *FrameEx* process.

Action: Verify the *Frame Exchange* gets activated; check the *Frame Exchange* error log to see if abnormalities have occurred.

Message: `get_poll_time_seed: 12070 : Error encountered on query for last status record, frame set <set-name> error msg is: <err-str>`

Description: An error was received from the frame set query routine for the frame set `<set-name>`. The `<err-str>` token provides the routine's assessment of error conditions.

Action: Evaluate the meaning of `<err-str>`, and correct any discrepancies. Verify the configuration of Frame Store; does the frame set exist with proper log file and permissions to allow needed access?

Message: `resend_queued_frames: 12072 : Encountered error on allocation of frame status structure, error msg is: None - UALLOC used for allocation`

Description: An error was received from the call to allocate memory for the process.

Action: Check that user resource limits have not been exceeded and a software error has not been introduced.

Message: `init_controller_exec: 12078: Unable to put self name in frame exchange command line`

Description: An error was received from the call to system routine `sprintf` in the building command.

Action: Check that a software error has not been introduced.

Message: `init_controller_exec: 12078 : Unable to put connection name in frame exchange command line`

Description: An error was received from the call to system routine `sprintf` in the building command.

Action: Check that a software error has not been introduced.

▼ Troubleshooting

Message: `init_controller_exec: 12080 : Unable to put fex
logging par file name in frame exchange command line`

Description: An error was received from the call to system routine *sprintf* in the building command.

Action: Check that a software error has not been introduced.

Message: `controller_exec_sig_catch: 12081 : Child
termination signal received, processing exit`

Description: A terminating signal was received from the operating system.

Action: None. The *Exchange Controller* will be restarted using the same method by which it was previously invoked. Check the error log for the *Frame Exchange* process for clues as to why the process terminated (*Frame Exchange* was child process).

Protocol Checker

Message: `main: Unable to open frame store for <store-path>,
err = <err-nbr>`

Description: An error was received from the routine attempting to open Frame Store at *<store-path>*. The routine's assessment of the error condition is coded in the *<err-nbr>* token.

Action: See error number in `libfs` for the meaning of *<err-nbr>*; correct problems based on the provided value. Verify that Frame Store files and directories exist and correct permissions are assigned.

Message: main: Unable to open frame set <set-name>,
 err = <err-nbr>

Description: An error was received from the routine attempting to open the frame set named <set-name>. The routine's assessment of the error condition is coded in the <err-nbr> token.

Action: The <set-name> is from the *Protocol Checker's* par file. Verify that the frame set name corresponds with definitions in the Frame Store configuration par file identified in the *framestore-parfile* par file parameter.

SOLVING COMMON PROBLEMS

To date, experience has shown that most problems encountered with the *CDS CD-1.1* occur at startup. This experience relates to configuration specification issues and a special class of configuration issues dealing with authentication.

CDS CD-1.1 software is executed with the intent of logging all abnormal and error conditions, which will include authentication and configuration conditions.

For further information on problem conditions, see ["Interpreting Error Messages" on page 43](#).

Error Recovery

CDS CD-1.1 software recovers from all but the most catastrophic errors without operator intervention. When a terminating condition is encountered by any *CDS CD-1.1* process, the program will write the reason for its termination to its log file. Log files should be examined to determine if the problem is likely to re-occur. Reoccurring exits are usually caused by either a configuration error or protocol violations by a protocol peer. In either case, the problem must be corrected to stop the reoccurring exit.

▼ Troubleshooting

A *CDS CD-1.1* process that has exited will be restarted by either an inbound connection request from a data provider or by the *ForeMan* process. The exception to this is termination of the *ForeMan* process, which does not have the ability to restart itself. If a *ForeMan* terminates, operator action is required. In such a case, examine the *ForeMan*'s log file, make adjustments as necessary, and restart the process as described in ["Operational Procedures" on page 15](#).

Catastrophic errors include the destruction of the Frame Store, frame log, and/or Disk Loop files. In the event that any of these files are destroyed the operational environment should be recreated as described in ["Installation Procedures" on page 105](#), and the system should be restarted as described in ["Operational Procedures" on page 15](#).

REPORTING PROBLEMS

The following procedures are recommended for reporting problems with the application software:

1. Diagnose the problem as far as possible.
2. Record information regarding symptoms and conditions at the time of the software failure.
3. Retain copies of relevant sections of application log files.
4. Whenever possible, collect a data set and configuration, which can be used as a test case to demonstrate the problem.
5. Contact the provider or maintainer for the software for problem resolution if local changes of the environment or configuration are not sufficient.

Chapter 4: Installation Procedures

This chapter provides instructions for installing the software and includes the following topics:

- [Preparation](#)
- [Executable Files](#)
- [Configuration Data Files](#)
- [Database](#)
- [Short Course for Installing Software](#)
- [Adding Data Providers](#)
- [Short Course for Adding Data Providers](#)
- [Adding Forwarding Destination](#)

Chapter 4: Installation Procedures

PREPARATION

Obtaining Released Software

The *CDS CD-1.1* software may be obtained via File Transfer Protocol (FTP) from a remote site or via a physical medium such as tape or CD-ROM. The software and associated configuration data files are stored as one or more tape archive (tar) files. The software and data files are first transferred via FTP or copied from the physical medium to an appropriate location on a local hard disk. The tar files are then untarred into a standard UNIX directory structure.

Hardware Mapping

The hardware on which to run the software components must be selected. Software components are generally mapped to hardware to be roughly consistent with the software configuration model. The hardware requirements scale with the number of data providers and consumers serviced by the data center and the number of authenticated data channels received.

UNIX System

CDS CD-1.1 User Account

A user account should be established for either the *CDS CD-1.1* or the monitoring system software. This user account is the name under which *CDS CD-1.1* applications will be started. Because the *CDS CD-1.1* software is designed to run primarily

in an automated fashion, such a definition aids in the management of system resources and control of the *CDS CD-1.1* processes.

Internet Daemon Configuration

Configuration changes addressed in this section require UNIX super user privileges.

Inbound connections by *CDS CD-1.1* data providers are serviced by the *Connection Manager* at the application level. To respond to a data provider request, the Internet daemon (*inetd*) of a selected contact machine must be configured to execute and pass control to the *Connection Manager*. The *Connection Manager* should execute on a highly available computer connected to the WAN.

IMS stations and NDCs should issue inbound connection requests to a well-known host and port number, for example, `spinach.pidc.org 8031`. After the *Connection Manager* host is determined, the following lines must be added to the `/etc/inetd.conf` file on the selected host:

```
## Added to support CDS, CD-1.1 connections
##
connmgr stream tcp nowait <name> /data/spinach/cds11/sbin/ run_idc_connmgr
```

The string `/data/spinach/cds11/sbin` should be replaced with whatever absolute path (*BINroot*) is selected for storing the *CDS CD-1.1* binaries, and `<name>` should be replaced with the user name under which the *CDS CD-1.1* software will execute.

Next, the `/etc/services` file must be updated. The Internet daemon monitors connection requests to UNIX port numbers and maintains a mapping between port numbers and services or the application responsible for responding to requests to use the ports. The port number selected for a service then becomes a "well-known" port number for the supporting application and is used by those processes that request service. *CDS CD-1.1* applications have been configured to use the port number 8031 as the well-known connection port for the *Connection Manager*. Edit the `/etc/services` file (on the selected host) by adding the following lines:

▼ Installation Procedures

```
## Added to support CDS, CD-1.1 connections
##
connmgr 8031/tcp          # CDS, CD-1.1 connection manager
```

The *Connection Manager Server* requires similar configuration definitions for the same reasons. The *Connection Manager* has been configured to use port number 8032 as the well-known connection port for *Connection Manager Servers*. Whereas the *Connection Manager* required changes on a single computer, the *Connection Manager Server* changes are needed on each computer used for servicing data from IMS stations or NDCs. On each host computer make the following edits:

Add the following lines to `/etc/inetd.conf`:

```
## Added to support CDS, CD-1.1 connections
##
connsrv stream tcp nowait <name> /data/spinach/cds11/sbin/ run_idc_connsrv
```

In `/etc/services` add the following lines:

```
## Added to support CDS, CD-1.1 connections
##
connsrv      8032/tcp          # CDS, CD-1.1connection server
```

Each of the modified/selected hosts must either have a restart signal sent to the Internet daemon or be rebooted for the changes to take effect.

The following command will issue the hang-up signal to the Internet daemon causing the Internet daemon to restart and reinitialize itself:

```
%kill -HUP <inetd-pid>
```

`<inetd-pid>` is the PID of the *inetd* process (it may be determined with the `ps` command).

File System

The *CDS CD-1.1* software uses three categories of directories for housing the binary executables, data stores, and logging. The physical location of each of these directories is independent. However, for reasons of processing efficiency, it may be advantageous to locate the directories on particular disk partitions or host comput-

ers. These issues will be addressed in subsequent paragraphs as necessary. Each of the categories of directories are referred to as a root and are given the following symbolic names:

- *BINroot* is the directory hierarchy root that has the binary executables.
- *RUNroot* is the directory hierarchy root that has the run-time data stores used by the software.
- *LOGroot* is the directory hierarchy root for log files written by the software.

These symbolic names are used by the software and in this document to refer to locations in the directory hierarchy. The definition of these names are provided by script files used to establish the run-time environment. See [“Creating Run-time Environments” on page 115](#) for additional information.

Firewall

The firewall of the data center will need to be modified to support connections and data flow from data providers as well as to data consumers.

Inbound Data Flows

Inbound connections will be directed to a Uniform Resource Locator (URL) and port number by data providers. Data providers (their IP address or URL as required by the firewall) will need to be registered. Firewall exceptions allow the connections to the well-known host and port number of the *Connection Manager* process. Each host that services connections to data providers will require an exception to allow connections from the data providers and are the hosts designated for executing the *Connection Manager Server* instances. *CDS CD-1.1* may be configured to impose either a strict or liberal mapping of providers to servicing hosts via entries to the `port_<inst>.par` files and **alphasite** and **dlman** DBMS tables (see [“Connection Manager/Connection Manager Server” on page 117](#)). A strict mapping will necessitate strict specification of firewall exceptions regarding the IP addresses and the ports of connecting sites. Alternatively, the software may be configured to allow the dynamic allocation of resources, that is, hosts servicing connections. This

▼ Installation Procedures

will lead to a different method for specifying firewall exceptions, where ranges of IP addresses are made available on multiple hosts. This second, more liberal approach is recommended and lends itself to higher availability of data flow.

Outbound Data Flow

Outbound data flows require a connection between the data provider host and a data consumer host. Data provider hosts are configured to execute a *Data Parser* process (see the paragraphs for the configuration of ["Data Center Manager" on page 133](#)). Firewall exceptions will be required between these hosts and the protocol servicing host or hosts of the data consumer.

EXECUTABLE FILES

A directory root should be selected to house the executable hierarchy of the *CDS CD-1.1* software. In making this decision, attributes of a NFS should be considered. When a NFS directory root is selected a globally visible directory is provided to all instances of the programs. This avoids multiple copies of software and attendant maintenance in update cycles. This advantage is countered by the fact that if the NFS partition host computer becomes unavailable, then all instances of the programs become inoperable. As an alternative, a directory root local to each machine hosting *CDS CD-1.1* applications may be selected. This configuration requires that multiple copies of software be installed. The advantage of this approach is that programs are only subjected to reliability concerns of the local computer. For high reliability it may be advantageous to select a local installation root. Descriptions in this document assume a single installation location. Generally, if a single installation point is not followed, the installation process would be repeated for each host selected for the *CDS CD-1.1* software. For example purposes `/data/spinach/cds11/bin` is used as the directory root for instructions. To establish such a directory enter the following command:

```
mkdir -p /data/spinach/cds11/bin
```

The `-p` option will create parent directories as necessary.

Move to the *BINroot* area using the change directory command:

```
cd /data/spinach/cds11/bin
```

Copy the executables from the build (or FTP, or untar) location to the *BINroot* directory.

DATA STORES

The run-time data stores are Frame Store and Disk Loop directories and files. Frame Stores and their component frame sets and frame logs are used heavily by the *CDS CD-1.1* and are an important part of the reliable data delivery design. As a result, these should be located on a computer that is highly reliable and not overburdened with other processing routines. Frame Stores at a data center have three principal users; Frame Stores are written predominately by the *Frame Exchanges* receiving data, read by *Data Parser* extracting time-series data, and read by *Frame Exchanges* forwarding data to other data centers. These uses should be kept in mind when identifying the installation location of the Frame Store. Disk-loop files contain parsed time-series data written by the *Data Parser*.

Log Files

Log files are used by all components of the *CDS CD-1.1* to capture both normal and abnormal occurrences of program execution. It is recommended that a log directory be defined on a partition residing on a machine where a given process will execute, but this is not a requirement. When the log location is common to multiple platforms, care should be taken to ensure that applications have unique log filenames to avoid logging collisions. For example, assume a computer hosts two *Frame Exchange* processes, one servicing a connection to station A and one servicing station B. If the logging location is identified as `/data/spinach/cds11/cds_logs` and the log files are both named `FE_log`, both *Frame Exchanges* will write output to the file `/data/spinach/cds11/cds_logs/FE_log`. The result will be difficult or impossible to interpret.

▼ Installation Procedures

Creating Stores—Logging, Frame Store, and Disk Loops

Scripts have been devised for creating the run-time environment needed by *CDS CD-1.1* at a data center. These scripts are contained in a utilities (`utils`) directory along with other subsystem tools. Sample contents of these scripts are provided in [“Appendix: Tools” on page A1](#). Prior to executing these setup scripts, adjustments should be made to the configuration to reflect data center requirements (see [“Configuration Data Files” on page 113](#) for additional information). The contents of the application configuration files must correspond with the utility scripts used to create the run-time environment and vice versa. The contents of configuration files in both areas should be reviewed prior to making any changes. The definition of the scripts for creating the run-time environment are dependent on the intended environment/configuration, and the intended environment/configuration must be correctly defined to run the *CDS CD-1.1*.

When the configuration of the system identifies disk partitions that only exist on a given machine or are local to a given machine, the setup script must be executed on that host. Assuming configuration changes have been made, change directories to be in the utilities/environment (`utils/env`) directory, and issue the following command to create the run-time environment:

```
setup_env idc
```

This script sets up some initial directories and variable names, and then calls a system-specific setup script (`setup_idc_env`). The system-specific scripts will finish the environment creation process. The `setup_env` script performs its work in a non-destructive manner. This aspect makes it particularly useful in the data center environment where it can be used to add entities to the environment without peril to existing definitions.

When executed, output lines report failure opening a frame log as follows:

```
Cannot open file: /<path string>/FrameStore/sg1.flog,  
error 2, No such file or directory
```

This is typical and is a result of the program used to create the frame sets. The user can verify that the script worked correctly by ‘`cd`’ing to the identified directory

and checking for the `.flog` file identified in the error output. If the file is *not* there, then something genuinely went wrong.

Disk Loop files required by the *Data Parser* for storing parsed time-series data are also created by the `setup_env` script. However, to correctly allocate and create these files, database tables describing sites and their data must be populated. See [“Data Parser” on page 123](#) for information on required tables and values.

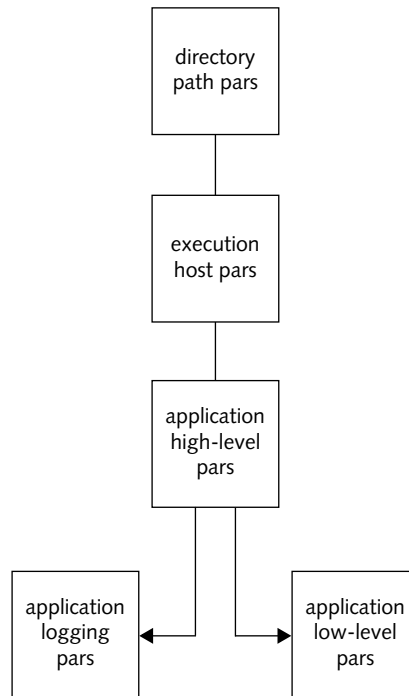
CONFIGURATION DATA FILES

Application Configuration Overview

Each *CDS CD-1.1* application process is configured with parameters in configuration parameter files (par) files. Nested par files have been used to implement a hierarchy of par files. The typical hierarchy has a directory and environment parameters at the top, followed by the application's parameters, followed by logging information and other lower-level application information. [Figure 6](#) shows this hierarchy, which is an abstraction of the par file structure. In practice, some par files have other nested definitions and some may omit a nested definition. In the omission case, parameters that would have been appropriate in the nested file are contained in the application's parameter file.

Definitions for directory paths within parameter files are given as relative paths. The root that ties the relative path to the physical directory structure used is defined by an environment variable that is set prior to execution of the software. The `cds_idc_env` shell script provides the definition of this and other needed environment variables; this file may be found in `<BINroot>/cds_idc_env`.

▼ Installation Procedures

**FIGURE 6. SAMPLE PAR FILE HIERARCHY**

A naming convention has been applied to the names of the par files used by *CDS CD-1.1* applications. The convention applies the name of the process as the first name element, followed by an instantiation element, and terminated with a `.par` suffix. For example, the *ForeMan* par file used to manage processes on host spinach would have the name, `ForeMan_spinach.par`.

Within par files the names of parameters are intended to provide insight into their purpose. The scope of parameters within each of the files is limited to the program that reads them. This can lead to the reuse of parameter names, and in some cases, meanings. In many cases comments in the par files describe the purpose of its parameters. Comments begin with a hash (#) character.

This section provides configuration information for those parameters that are considered likely to require site- or installation-specific alteration. More parameters may be “configured” than those identified here. Included in the parameter files are parameters that are only of interest to software maintenance engineers.

All the *CDS CD-1.1* processes have the ability to write processing information to a log file. The amount of information written is controlled by a configuration parameter read by the program from one of its par files. This parameter is referred to as the log level. Typically, the log level for software would be set to minimize the amount of logging output from the program. However, to investigate the behavior of a program, the log level can be altered to increase the verbosity of a program. Log levels range in value from 1–10, where a higher value indicates increased verbosity. Log level 1 represents fatal error conditions; these are always logged. Additionally, by convention, the software will also issue a request to update the operating system's system log if a fatal condition is detected.

As identified in [“Creating Stores–Logging, Frame Store, and Disk Loops” on page 112](#), the contents of the application configuration files must correspond with the utility scripts used to create the run-time environment and vice versa. The contents of configuration files in both areas should be reviewed prior to making any changes. The definition of the scripts for creating the run-time environment are dependent on the intended environment/configuration, and the intended environment/configuration must be correctly defined to run the *CDS CD-1.1*.

Creating Run-time Environments

cds_idc_env is a shell script file for creating a run-time environment for the *CDS CD-1.1*. This file is located in the binary installation directory, for example, */data/spinach/cds11/bin*. Within this file is the variable *CMSroot*, which specifies a root directory where elements of the software may be found. *CMSroot* is provided to drive a common location for the *CDS CD-1.1* software elements. This is the recommended approach. However, if it is not possible to define a single directory root for all elements of the software, separate definitions may be provided for each class of directories (*BINroot*, *RUNroot*, *LOGroot*) in this file to accommodate different roots.

▼ Installation Procedures

The following definitions in this file should be checked and updated as necessary.

- The definition of the environment variable *CMSroot* should identify the installation root of the software, for example, */data/spinach/cds11*.
- The *CONTINUOUS_DATA-DIR* parameter should identify the location where the run-time configuration files of the *CDS CD-1.1* processes are located. Under this directory will be directories for *CDS CD-1.1* components containing par files. For example, building on the *CMSroot* definition above, this variable would be set to *\$CMSroot/config*.

Subsystem-level Configuration

[Figure 6 on page 114](#) shows that the *cds.par* file is located at the directory path level. This parameter file provides high-level definitions needed by the *CDS CD-1.1* and may be found in the system directory in the configuration hierarchy, for example, *<CMSroot>/config/system_specs*. Check the definition of the following items, and update them as necessary:

- Set the parameter or parameters *IDC<n>name* to identify the computers that are part of the IDC environment, for example, *IDC1name=spinach.cmr.gov*. By convention the *IDC1name* variable identifies the host that data providers would contact to establish a data flow. Other host computers (*IDC2name*, *IDC3name*, and so on) identify other hosts designated for *CDS CD-1.1* use.
- Set the parameter or parameters *IDC<n>addr* to the IP addresses corresponding to computers identified in the parameters *IDC<n>name*. Addresses are specified in dot notation.
- Set the parameter(s) *NDC<n>name* to identify the computer(s) to be contacted when establishing forwarding data flow(s). For example, if *CDS CD-1.1* data are to be forwarded to the URL *big_rock.ndc.gov* as the contact host (in the same way that *IDC1name* was defined), then *NDC1name= big_rock.ndc.gov* would be assigned.
- Set the parameter *BINroot* to identify the directory where the *CDS CD-1.1* binaries have been placed.

- Set the parameter *RUNroot* to the directory hierarchy root where the *CDS CD-1.1* run-time files will be written and accessed. This would be the root for the Frame Store and Disk Loops. When *Frame Exchanges* are on different hosts, this directory must be NFS mounted.
- Set the parameter *LOGroot* to the directory hierarchy root where the *CDS CD-1.1* run-time log files will be written. Beware of filename collisions if this directory is NFS mounted.

Connection Manager/Connection Manager Server

The *Connection Manager* and the *Connection Manager Server* process are started via the Internet daemon on their execution host. They are also configured with parameter files located in the application configuration directory hierarchy, *<CONTINUOUS_DATA-DIR>/ConnMgr*.

In the file *connmgr_idc.par*:

- The parameter *well_known_port* is set to the value of an available port assignment on the host computer. This value is currently set to a value of 8031. If this value is changed, the same change must be made in the */etc/services* file on the host computer.
- Set the parameter *database* to the user name and password string for the database to be used in looking up acceptable connection requestors. Typically an installation would have a system-wide environment variable or macro for holding the user name and password pair. This simplifies maintenance and limits the propagation of passwords, which would compromise security.

In the file *connsvr_idc.par*:

The parameter *well_known_port* is set to the value of an available port assignment on the host computer. This value is currently set to 8032. If this value is changed, an identical substitution must also be substituted in the */etc/services* file on the host computer.

▼ Installation Procedures

The `port_idc.par` file provides definitions that specify the UNIX ports to be used for socket communications with the data providers supported by the host computer. Entries in this table must coincide with entries in the **alphasite** and **dlman** database table for identifying supported data providers and their server host. There must be an entry in this file for every data provider supported by the *Connection Manager Server* reading this file.

In the file `port_idc.par`:

1. Set the provider name into the parameter by adding its name after the string "port-" and before the character "=". For example, the line for identifying station CD02 would have the following definition: `port-CD02=12345`.
2. Set the number of the UNIX port to use for communication to the specified provider. Assignment of UNIX port numbers is generally coordinated or approved by the system administrator.

The **alphasite** database table must contain an entry for each data provider recognized by the *Connection Manager*. The following discussion addresses entries required for a single data provider. Similar alterations would be needed for additional data providers. Assuming that the user has entered a SQL*Plus session, the following commands can be used to add database table entries. Use the following command to add an entry to the **alphasite** table:

```
SQL> insert into alphasite (sta, address, prefdlid)
      values ('<sta>', '<sta-addr>', <id-val>);
SQL> update alphasite set time=0 where sta='<sta>';
```

In the above commands `<sta>` is replaced with the ASCII name of the data provider, `<sta-addr>` is replaced with the dot-notation IP address of the data provider, and `<id-val>` is replaced with an index value of an entry in the **dlman** table, which is also used for establishing connections. The `update` command is used to initialize a time value that represents the last time a connection request was received from the data provider. The following example shows the two commands with substitutions:

```
SQL> insert into alphasite (sta, address, prefdlid)
      values ('CD02', '140.162.3.133', 10);
SQL> update alphasite set time=0 where sta='CD02';
```

To add an entry to the **dlman** table:

```
SQL> insert into dlman (dlid, machine, running,  
    connmanport) values (<id-val>, '<server-host>', 'n',  
    <port-id>);
```

In the above command *<id-val>* is replaced with a unique integer identifier for this table entry. The table entry represents an available *Connection Manager Server*. *<server-host>* is replaced with the ASCII name of the *Connection Manager Server* host computer name that will service requests for all **alphasite** entries that designate this **dlman** (via the *<id-val>*) entry; and *<port-id>* is replaced by the integer value of the UNIX port used to contact this server. The value provided for *<port-id>* must match the value supplied for the parameter *well_known_port* in the *connsvr_idc.par* file. An example of the above command with substitutions is as follows:

```
SQL> insert into dlman (dlid, machine, running,  
    connmanport) values (10, 'bat', 'n', 8032);
```

Connection Originator

The *Connection Originator* is a data provider process. In the context of the IDC, the *Connection Originator* will be used for forwarding data to subscribing NDCs. The *Connection Originator* is configured via the file *ConnOrig.par*. The *Connection Originator* supports the use of two command line parameters: *self* and *connection*. These two arguments are used to substitute parameters of the process's par file and specifies who is making the connection and to whom, respectively. If these parameters are not used, a separate par file is required for each outbound connection. Additionally, there is a cascading effect on the use of the *self* and *connection* arguments. Use of these parameters enables the *Connection Originator* to symbolically pass these arguments to the *Exchange Controller*, where the effect on the number of separate par files needed is the same. The *ConnOrig.par* file is located in the *<CONTINUOUS_DATA-DIR>/ConnOrig* directory. In this file:

▼ Installation Procedures

1. Set the *co-connection-manager-ports* variable to the hostname and port number of the *Connection Manager* to contact for a connection, for example:

```
co-connection-manager-ports=big_rock.ndc.gov: 8031
```

Alternatively, using variable substitution, a value of *NDC1* for the variable *connection* and the *NDC1name* definition provided in *cds.par*, this assignment could be stated as:

```
co-connection-manager-ports="$( $(connection)lname ): 8031."
```

The port number must be the port number of the *well_known_port* of the *Connection Manager* at the remote data center.

2. Set the variable *co-exch-ctl-pars* to the *par* assignment string for the specific *par* file supporting the instance of the *Exchange Controller* servicing the requested connection. When the *connection* variable is not used, a different *par* file is needed for each instance of the *Exchange Controller*. An example using variable substitution follows:

```
co-exch-ctl-pars="self=$(self) connection=$(connection)
par=$(CONTINUOUS_DATA-DIR)/ExCtrlr/ ctlr_fex_$(self)_$(connection).par"
```

3. Set *co-station-name* to the site name of this *Connection Originator*. For a forwarding IDC connection this parameter would receive the assignment, IDC, or preferably the assignment of the variable *self*.
4. Set *co-creator* to the site name of this *Connection Originator*, that is, typically to the same value as *co-station-name*. This item will be used to look up the site's private key for digitally signing protocol frames. As a result this must be the exact name as registered in the index file for private keys. See ["Authentication" on page 134](#).

Frame Exchange/Exchange Controller

The *Frame Exchange* and the *Exchange Controller* are a processing pair that handle the transport of frames from one *CDS CD-1.1* platform to another. Because the processes have many of the same configuration requirements, their configuration

is contained in a single par file. In general, parameters in the par file read by the *Exchange Controller* begin with the characters "ctlr," while those read by the *Frame Exchange* have the prefix "fex." *FrameExchange/Exchange Controller* processing pair is instantiated for each connection to and from a CDS CD-1.1 platform. The exchange/controller pair is responsible for transmitting and receiving frames for a configured list of frame sets (which may be a list of one). The configuration for a specific instance of the *FrameExchange/Exchange Controller* pair is provided by a file named `ctlr_fex_<instance>.par`, where *<instance>* is a self connection description. As discussed in the ["Connection Originator" on page 119](#), this usage takes advantage of cascading variables and variable substitution. For example, the par file handling the connection instance between the IDC and provider stations would be found in the directory `<CONTINUOUS_DATA-DIR>/ExCtlr` and named `ctlr_fex_idc_sta.par`. In this file:

1. Set *ctlr-polling-frame-set-list* parameter to a double-quoted, comma-separated list of frame sets that this controller will poll for frames that need to be sent. The names in this list must match exactly the strings used to define frame sets in the Frame Store configuration par file (see ["Frame Store" on page 129](#)). This list controls those frames that will be polled for frames to be sent by the exchange/controller pair. For outbound connection instantiations this list is required because without it, no frames will be sent. For inbound instances the list definition is optional and typically will not be defined.
2. Set *ctlr-writing-frame-set-list* to a double-quoted, comma-separated list of frame sets. Typically, this will be a list of one identifying the *Exchange Controller's* command frame set that is only used to store Alert Frames created by the controller.
3. Set *ctlr-reading-frame-set-list* to a double-quoted, comma-separated list of frame sets that the controller will read for inbound commands and communications.

▼ Installation Procedures

4. Set *ctrl-logging-frame-set-list* to a single frame set name. The *Exchange Controller* will use a frame set and Frame Store library to keep track of frames queued for sending and those that have been acknowledged by the destination recipient. This name provides the identification of that frame set.
5. Set *ctrl-fs-par-file* to identify the Frame Store configuration par file that describes the Frame Store and frame sets. (See ["Frame Store" on page 129](#).) The frame set names of this par file must correspond with the definitions in the exchange/controller par file.
6. Set *ctrl-cmd-frame-set* to one of the frame set names specified in the *ctrl-writing-frame-set-list*. This is the frame set that will be used by the *Exchange Controller* when it needs to create an Alert Frame.
7. Set *ctrl-cmd-src-id* to the character string that defines the *self* side of the *Exchange Controller's* connection. This parameter is used in creating command frames. For example, at the IDC this parameter would contain the value IDC. At an IMS station the parameter would have the name of the station, for example, CD02 if the station were named CD02.
8. Set *ctrl-cmd-dest-id* to the character string that defines the destination for command frames created by the *Exchange Controller*. For example, for the connection between the IDC and NDC-BB, this parameter would contain the value NDCBB.
9. Set *fex-fs-par-file-name* to the Frame Store configuration par file provided to the *Frame Exchange*. This parameter must be set to either the *ctrl-fs-par-file* parameter itself or the same file as provided for that parameter's definition.
10. Set *fex-reading-frame-set-names* to be the list of frame sets read by *Frame Exchange* for sending. This list will include the controller's command frame set and the list of frame sets polled by the *Exchange Controller* (*ctrl-polling-frame-set-list*).
11. Set *fex-writing-frame-set-names* to the frame set to use for inbound frames. This list will include the frame set for the controller commands and communications (*ctrl-reading-frame-set-list*).

Both the *Frame Exchange* and the *Exchange Controller* use multiple-level par files for defining logging parameters. The first level of par files provides common definitions that are used for all instantiations of the exchange or controller that can “see” the common file. The second level provides instantiation-specific definitions for the respective application. This approach facilitates changes to globally affect logging and alter logging related to a specific instance of a program. In this scheme, global definitions are read first followed by the specific instance definitions. Hence, specific instance definitions will override global definitions.

The *Exchange Controller* will use the Frame Store to track those frames sent to a peer exchange/controller pair. This requires a frame set definition in the Frame Store for each instance of the *Exchange Controller* at a given site. The definition of the controller’s “special” frame set is contained in the overall definition of the Frame Store at a given site. See [“Frame Store” on page 129](#) for additional information.

Data Parser

The *Data Parser* translates time-series data received in protocol frames to the format used by signal processing and analysis software. *Data Parser* must be configured with database tables, run-time parameter values, and the Disk Loop files. There may be more than one instance of *Data Parser* at a data center. Each instance of the *Data Parser* requires run-time configuration files for defining unique parameter values. The *Data Parser* uses two par configuration files, which incorporate the instance ID into their name: `dl<inst>.par` and `auth_<inst>.par`. The `auth_<inst>.par` file provides authentication parameters; and the `dl<inst>.par` file provides all other parameters. These par files are in the directory `<CONTINUOUS_DATA-DIR>/DLParse`. Configure these par files as follows:

In the file `auth_<inst>.par`:

1. Set *auth-enabled* to indicate whether Channel Subframes should be authenticated (assuming a key is identified in the subframe). A value of zero indicates no authentication, and a value of one invokes authentication.

▼ Installation Procedures

2. Verify that *auth-path* defines the directory root used for housing authentication certificates and keys. See [“Authentication” on page 134](#).

In the file `d1<inst>.par`:

1. Set the parameter *dlid* to represent the instance of the *Data Parser*.
2. Set the parameter *database* to the user name and password string for the database used by the *Data Parser*. Typically an installation would have a system-wide environment variable or macro for holding the user name and password pair. This simplifies maintenance and limits the propagation of passwords.
3. Set *sta_fs_list* to the double-quoted, space-separated list pair providing stations and frame sets to be processed by the *Data Parser*. For example, if data from stations CD01 and CD02 are to be processed and their Data Frames are in the frame sets CD01:0 and CD02:0, then the following string would be supplied: “CD01 CD01:0 CD02 CD02:0”.
4. Set *framestore_config* to specify the Frame Store configuration file that defines the Frame Store at the data center. Frame sets specified in *sta_fs_list* must be part of the configuration contained in the file referenced by this parameter.
5. Set *machine* to the name of the computer on which the *Data Parser* will be executing.

The *Data Parser* uses the Frame Store not only to retrieve Data Frames to be processed, but also to store information about frames processed, and authentication results. In support of storing its own information, the *Data Parser* requires the definition of frame sets in the Frame Store. Two frame set definitions are required for each site whose data are to be processed by the *Data Parser*. See [“Frame Store” on page 129](#).

The *Data Parser* will use the following database tables: **affiliation**, **alphasite**, **dlfile**, **instrument**, **sensor**, **sitechan**, **wfconv**, **wfdisc**, and **wfproto**. These tables are part of the IDC database schema [\[IDC5.1.1Rev2\]](#) so no tables need to be created. However, data values will need to be added to some of these tables. Create a Structured

Query Language (SQL) script for making the updates. For explanation purposes, the following paragraphs provide the necessary commands for configuration, as if the user were logged into a SQL*Plus interactive session.

The **instrument** table provides data about the types of sensor instruments used by stations to gather signal data. Additions to the **instrument** table are not common. However, if a new instrument type comes into service, information about it is required for the parser (and other software) to operate correctly. As an example, the following command would be used to insert a new entry in the **instrument** table:

```
SQL> insert into instrument (inid, insname, instype,
band, digital, samprate, ncalib, ncalper, dir,
dfile, rsptype, lddate) values (50000, 'Data
Simulator', 'dummy_type', 's', 'd', 40, .01, 1,
/cmss/config/station_specs/rsp, S-13.sp, 'paz',
sysdate);
```

The **site** table defines the sites that will be delivering CDS CD-1.1 data to the IDC. An entry in this table describes the sensor's site. When a sensor array is providing data, a table entry is needed to represent the aggregate "array" as well as one entry for each site within the array. As an example, the following commands would be used to establish a three-sensor array where the name of the array is CD02 and the names of the sites are S2A0, S2A1, and S2A2:

```
SQL> insert into site (sta, ondate, offdate, lat,
lon, elev, staname, statype, refsta, dnorth,
deast, lddate) values ('CD02', 2000001, -1,
62.4932, -114.6053, 0.197, 'DummyStation CD02',
'ar', 'CD02', 0.000, 0.000, sysdate);

SQL> insert into site (sta, ondate, offdate, lat,
lon, elev, staname, statype, refsta, dnorth,
deast, lddate) values ('S2A0', 2000001, -1,
62.4932, -114.6053, 0.197, 'DummyStation CD02',
'ss', 'CD02', 0.000, 0.000, sysdate);
```

▼ Installation Procedures

```
SQL> insert into site (sta, ondate, offdate, lat,
lon, elev, staname, statype, refsta, dnorth,
deast, lddate) values ('S2A1', 2000001, -1,
62.4932, -114.6053, 0.197, 'DummyStation CD02',
'ss', 'CD02', 0.000, 0.000, sysdate);

SQL> insert into site (sta, ondate, offdate, lat, lon,
elev, staname, statype, refsta, dnorth, deast,
lddate) values ('S2A2', 2000001, -1, 62.4932, -
114.6053, 0.197, 'DummyStation CD02', 'ss', 'CD02',
0.000, 0.000, sysdate);
```

The **affiliation** table provides a mapping between a site and the reference station named. When a site is part of an array the sensor site and the reference station will generally be different, but for a single sensor station they are one and the same. *CDS CD-1.1* processing requires an **affiliation** definition for all data providers, including the single sensor stations. The following command would be used to add **affiliation** entries. An **affiliation** entry is required for each site reported by a reference station:

```
SQL> insert into affiliation (net, sta, lddate) values
('CD02', 'S2A0', sysdate);
SQL> insert into affiliation (net, sta, lddate) values
('CD02', 'S2A1', sysdate);
SQL> insert into affiliation (net, sta, lddate) values
('CD02', 'S2A2', sysdate);
```

The **sensor** table contains entries that describe the sensors installed at the stations. Each row in the table describes a data source. As an example, the following three commands would be used to describe data from a three-channel (she, shn, and shz) site:

```
SQL> insert into sensor (sta, chan, time, endtime, inid,
chanid, jdate, calratio, calper, tshift, instant,
lddate) values ('S2A0', 'she', 946684800.000,
9999999999.999, 50000, 120123, 2000001, 1, 1, 0, 'y',
sysdate);
```

```
SQL> insert into sensor (sta, chan, time, endtime, inid,
chanid, jdate, calratio, calper, tshift, instant,
lddate) values ('S2A0', 'shn', 946684800.000,
999999999.999, 50000, 120124, 2000001, 1, 1, 0,
'y', sysdate);
```

```
SQL> insert into sensor (sta, chan, time, endtime, inid,
chanid, jdate, calratio, calper, tshift, instant,
lddate) values ('S2A0', 'shz', 946684800.000,
999999999.999, 50000, 120125, 2000001, 1, 1, 0,
'y', sysdate);
```

The **sitechan** table provides the definitions for sensor channels for a sensor at a given site. An entry is needed in this table for each channel of data being provided in the *CDS CD-1.1* format. As an example, the following commands would be used to establish a site named S2A0, providing three channels of signal data (she, shn, and shz):

```
SQL> insert into sitechan (sta, chan, ondate, chanid,
offdate, ctype, edepth, hang, vang, descrip, lddate)
values ('S2A0', 'she', 2000001, 120123, -1, 'n',
.01, 90.0, 90.0, 'short-period e-w', sysdate);
```

```
SQL> insert into sitechan (sta, chan, ondate, chanid,
offdate, ctype, edepth, hang, vang, descrip, lddate)
values ('S2A0', 'shn', 2000001, 120124, -1, 'n',
.01, 90.0, 0.0, 'short-period n-s', sysdate);
```

```
SQL> insert into sitechan (sta, chan, ondate, chanid,
offdate, ctype, edepth, hang, vang, descrip, lddate)
values ('S2A0', 'shz', 2000001, 120125, -1, 'n',
.01, 0.0, 0.0, 'short-period vertical', sysdate);
```

The **wfconv** table is used to provide data necessary for converting waveforms contained in protocol frames into a format that can be exploited by analysis and signal processing software. Information in this table includes signal data type, compression scheme, and sample rate. Like the **sitechan** table, this table requires an entry for each channel of the *CDS CD-1.1* data. As an example, the following commands would be used to establish site S2A0, which provides three channels of signal data:

▼ Installation Procedures

she, shn, and shz. The *y* designation corresponding to the *inauth* field indicates that Channel Subframe authentication will take place. To disable authentication processing, a value of *n* would be provided.

```
SQL> insert into wfconv (sta, chan, chanid, inauth,
    incomp, intype, insamp, outauth, outcomp, outtype,
    outsamp, strip, commid, lddate) values ('S2A0',
    'she', 120123, 'y', '-', 's4', 400, 'n', '-', 's3',
    400, 'y', -1, sysdate);

SQL> insert into wfconv (sta, chan, chanid, inauth,
    incomp, intype, insamp, outauth, outcomp, outtype,
    outsamp, strip, commid, lddate) values ('S2A0',
    'shn', 120124, 'y', '-', 's4', 400, 'n', '-', 's3',
    400, 'y', -1, sysdate);

SQL> insert into wfconv (sta, chan, chanid, inauth,
    incomp, intype, insamp, outauth, outcomp, outtype,
    outsamp, strip, commid, lddate) values ('S2A0',
    'shz', 120125, 'y', '-', 's4', 400, 'n', '-', 's3',
    400, 'y', -1, sysdate);
```

The **wfproto** table provides waveform prototype information used by the *Data Parser* in the conversion of time-series data. Like the **sitechan** and **wfconv** tables, **wfproto** requires an entry for each channel of *CDS CD-1.1* data. As an example the following commands would be used to establish a site named S2A0, providing three channels of signal data (she, shn, and shz):

```
SQL> insert into wfproto (sta, chan, time, wfid, chanid,
    jdate, endtime, nsamp, samprate, calib, calper,
    instype, segtype, datatype, clip, dir, dfile, foff,
    commid, lddate) values ('S2A0', 'she',
    946684800.000, -1, 120123, 2000001, 9999999999.999,
    0, 40, .01, 1, 'NOTYPE', '-', 's3', '-', '-', '-',
    0, -1, sysdate);

SQL> insert into wfproto (sta, chan, time, wfid, chanid,
    jdate, endtime, nsamp, samprate, calib, calper,
    instype, segtype, datatype, clip, dir, dfile, foff,
    commid, lddate) values ('S2A0', 'shn',
```

```

946684800.000, -1, 120124, 2000001, 9999999999.999,
0, 40, .01, 1, 'NOTYPE', '-', 's3', '-', '-', '-',
0, -1, sysdate);

SQL> insert into wfproto (sta, chan, time, wfid, chanid,
jdate, endtime, nsamp, samprate, calib, calper,
instype, segtype, datatype, clip, dir, dfile, foff,
commid, lddate) values ('S2A0', 'shz',
946684800.000, -1, 120125, 2000001, 9999999999.999,
0, 40, .01, 1, 'NOTYPE', '-', 's3', '-', '-', '-',
0, -1, sysdate);

```

The *Data Parser* requires that Disk Loop directories and initial Disk Loop files exist prior to execution. The establishment of these entities is only required for the initial execution, unless they are somehow deleted; subsequent executions will utilize existing definitions. Disk Loop files are needed for each channel of data from each sensor and site. The database tables used by the *Data Parser* are also needed to create Disk Loop files. These tables must be populated prior to execution of the script(s), which establish the run-time environment. See [“Appendix: Tools” on page A1](#) for additional information on the environment creation script.

Frame Store

The Frame Store is a durable store used to house the CD-1.1 protocol frames. Frame Stores exist where frames are sent or received. The Frame Store is not a process, but rather a directory structure and files. The Frame Store is configured with a parameter file. Generally, a given site should have only one Frame Store par file, which makes managing store definitions less cumbersome. A UNIX man page exists for the Frame Store configuration as the man section 4 entry, *FrameStore*. Refer to the man page for information about the par file. The Frame Store configuration file is found in the directory <CONTINUOUS_ DATA-DIR>/FrameStore with a filename similar to `fstore_idc.par`. A Frame Store par file should have frame set definitions (entries) for each of the following:

- The main data/frame supply from each station/data center providing time-series data. A convention has been adopted to name the time-series frame set <site-name>:0.

▼ Installation Procedures

- From the frame provider to the frame consumer. This will be in addition to the frame set used for the main time-series frame supply. A second adopted frame set naming convention uses names *<creator>:<destination>*. For example, a frame set for station CD02's connection to the IDC would be defined as CD02:IDC. This same name can and must be used at both the IDC and the station CD02.
- To the frame provider from the frame consumer. In the previous example this results in a frame set named IDC:CD02. This complies with the convention *<creator>:<destination>*, as this set will store frames sent from the consumer to the provider.
- A frame set for each *Exchange Controller* instance at the site. Slight liberty with the creator/destination convention is used for these sets; they are named CTLR:*<creator-destination>*. Using the IDC connection to site CD02 as an example, the set would be named CTLR:IDC-CD02.
- A frame set for each site providing data to be processed by the *Data Parser*, that is, one for each *<station-name>:0* set. This set must have a name of the form *<station-name>:DLPARSE*. Following the previous example, this would result in the name CD02:DLPARSE.
- A frame set for each site providing signed Channel Subframes to be processed by the *Data Parser*. This set must have a name of the form *<station-name>:AUTH*, for example, CD02:AUTH.

Segments of a Frame Store par file containing the above definitions would be similar to the following:

```

#!BeginTable ChanGroup
|   name   | id | MinDur | MaxDur | FrameSz | NextFac | InitFrames | BinDur | LogEnt | TextSz
.
CD02:0      4      600      3600      22000      1.00      100      600      20000      256
CD02:IDC    5      600      3600      800      1.00      100      600      20000      256
IDC:CD02    6      600      3600      800      1.00      100      600      20000      256
.
.
.
CTLR:IDC-CD02 201      0      0      0      1.00      100      600      20000      256
.
.
.
CD02:DLPARSE 1004      0      0      0      0      0      0      0      20000      0
CD02:AUTH 1005      0      0      0      0      0      0      0      20000      0
#!EndTable

.
.
.
Channels[4]="S2A0/shz S2A0/shn S2A0/she S2A1/shz S2A1/shn S2A1/she S2A2/shz S2A2/shn S2A2/she
FrameLog[4]="${RUNroot}/FrameStore/cd02.flog"
Channels[5]="XX/sz XX/se XX/sn"
FrameLog[5]="${RUNroot}/FrameStore/cd02_idc.flog"
Channels[6]="XX/sz XX/se XX/sn"
FrameLog[6]="${RUNroot}/FrameStore/idc_cd02.flog"
.
.
.
Channels[201]="XX/sz XX/se XX/sn"
FrameLog[201]="${RUNroot}/FrameStore/ctlr_idc-CD02.flog"
.
.
.

```

```
Channels[1004]=$(Channels[4])
Framelog[1004]=$(RUNroot)/FrameStore/dlp_cd02_parse.flog
Channels[1005]=$(Channels[4])
Framelog[1005]=$(RUNroot)/FrameStore/dlp_cd02_auth.flog
.
.
.
#!/BeginTable CD02:0
| VolumeId | Dir | MaxSize
| 0 | $(RUNroot)/FrameStore | 13200
#!/EndTable
#
#!/BeginTable CD02:IDC
| VolumeId | Dir | MaxSize
| 1 | $(RUNroot)/FrameStore | 500
#!/EndTable
#
#!/BeginTable IDC:CD02
| VolumeId | Dir | MaxSize
| 1 | $(RUNroot)/FrameStore | 500
#!/EndTable
.
.
.
#!/BeginTable CTLR:IDC-CD02
| VolumeId | Dir | MaxSize
| 1 | $(RUNroot)/FrameStore | 500
#!/EndTable
```

Data Center Manager

The *Data Center Manager (ForeMan)* is a control process used to start, monitor, and restart processes in the CDS CD-1.1. In particular, the *Data Center Manager* controls those processes that are not started by external stimuli. This excludes those processes invoked via a data provider's connection request, that is, *Connection Manager/Connection Manager Server* and inbound *Exchange Controller/Frame Exchange* pairs. At the IDC this leaves, at a minimum, the *Data Parser(s)* and the *Data Parser* for outbound (forwarding) data connections. A *Data Center Manager* instance executes on each host computer and that executes the CDS CD-1.1 processes (according to the above description). Each manager requires its own configuration to identify specific instances of other processes. The *ForeMan* configuration par file is in the directory `<CONTINUOUS_ DATA-DIR>/ForeMan` with a name of `ForeMan_<inst>.par`. Within the par file, job control templates are defined for each process that the manager is to control. In the `ForeMan_<inst>.par` file:

1. Define *fm-job-template-ids* to be the space-separated list of job templates defined in the par file where there is one template for each job to be controlled. For example, if two job templates are to be defined, then this parameter might have the definition `ConnOrig DLParse` relating to two templates with these names as their discriminator.
2. Provide job template definitions. The names specified in *fm-job-template-ids* are used in parameter names within the template. Continuing with the previous example, the first template parameter is `fm-<job>-exec-path`; this becomes `fm-ConnOrig-exec-path`. Template definitions are as follows:
 - `fm-<job>-exec-path`
Provide the command line string for invoking the process. This would include a fully qualified path name and command line arguments as appropriate.
 - `fm-<job>-min-runtime`
Time threshold in seconds for determining if process was successfully executed/started (by *ForeMan*).

▼ Installation Procedures

- `fm-<job>-max-restarts`
Number of times to restart a process when an invocation fails: process runs less than `fm-<job>-min-runtime` seconds.
 - `fm-<job>-restart-waits`
The comma-separated list of `fm-<job>-max-restarts` times, in seconds, which provides the wait times between successive restarts of a failed process. If a process executes successfully (longer than `fm-<job>-min-runtime` seconds) the manager will restart it according to its `fm-initial-event` profile.
 - `fm-<job>-command-on-failure`
The command to execute after `fm-<job>-max-restart` failures to run a process. This references an entry in the `fm-initial-events` list.
 - `fm-<job>-command-and-control`
A zero (false) or one (true) indication of whether this job will provide control information to the *ForeMan* via a socket connection.
3. Define `fm-initial-events` to provide the list of jobs the *Data Center Manager* is to start. The definition provides the number of seconds to wait before starting a job (either initially, or after a successful execution) and the name of a job template from `fm-job-template-ids`.

Authentication

Authentication configuration is needed to verify signed frames and Channel Subframes and to sign frames. If an automated Certificate Authority (CA) capability is being provided to the system, the description provided below may not be pertinent based on the capabilities provided by the CA. Under the configuration directory an `auth` directory contains the `sensor_pub`, `sensor_pvt`, and `CACert` subdirectories. These would be found at `<CONTINUOUS_DATA-DIR>/auth`. The `sensor_pub` directory must be updated to contain the public key files for all sites providing signed frames that are to be received/validated by this site. Additionally, an index file `index.txt` in this directory must be updated to describe the new key. Each entry in the index file specifies:

- public key file's name

- entity belonging to the key (for example, a sensor-channel for signed Channel Subframes)
- 0 (false) or 1 (true) specifying whether frames or Channel Subframes from the entity should be authenticated
- serial number/integer index value unique to this file
- not before date and time
- not after date and time

The file containing the private key for a signing site should be placed in the `sensor_pvt` subdirectory. The `sensor_pvt` directory, like `sensor_pub`, contains an `index.txt` file, which must be updated. The format of this index file is exactly like that of the `sensor_pub` directory and similarly requires a line for each key in the directory.

The `CACert` directory currently holds the certificates of the CAs that issue the certificates used to sign frames at the site. If a new CA certificate is generated or received, place it in this directory.

DATABASE

This section describes database elements required for operation of this software component, including accounts and tables.

Accounts

In the *CDS CD-1.1* software suite the *Connection Manager* and the *Data Parser* are the only elements that use the DBMS. No special database accounts other than standard user access are required by these processes.

Tables

[Table 1](#) contains DBMS tables and the associated *CDS CD-1.1* software components that use them.

▼ Installation Procedures

TABLE 1: DATABASE TABLES AND CDS CD-1.1 APPLICATIONS

Table Name	CDS CD-1.1 Process
affiliation	<i>Data Parser</i>
alphasite	<i>Connection Manager</i>
dlman	<i>Connection Manager</i>
instrument	<i>Data Parser</i>
sensor	<i>Data Parser</i>
site	<i>Data Parser</i>
sitechann	<i>Data Parser</i>
wfconv	<i>Data Parser</i>
wfproto	<i>Data Parser</i>

See the configuration descriptions for the processes in [Table 1](#) and [“Adding Data Providers” on page 140](#) for additional information about table contents and configuration.

SHORT COURSE FOR INSTALLING SOFTWARE

This section provides an abbreviated list of activities required to install a release of *CDS CD-1.1* software. These activities assume that a tape archive (tar) file containing the software is on the installation system and that the tar file was created on a system with the same directory hierarchy/configuration as the system where the installation is to take place. This will ensure that when the tar file is “untarred” directories and files will be placed in their correct hierarchical location.

Shutdown of Running CDS CD-1.1

To shut down a running CDS:

1. Rename the *Connection Manager* run script to prevent further inbound connections:

```
cd <root-path>/bin
mv run_idc_connmgr run_idc_connmgr.sav
```

2. Get a process list on each host servicing an outbound connection or a *Data Parser*; then issue a command to kill the *ForeMan* processes:

```
rlogin <host> -l <cds-name>
ps -uf cmss
kill <ForeMan-pid(s)>
```

Wait for approximately 2 minutes for *ForeMan*-controlled processes to terminate.

3. Get a process list on each host servicing an inbound connection; then issue a command to kill the *Exchange Controller* processes:

```
rlogin <host> -l <cds-name>
ps -uf cmss
kill <ExCtlr-pid(s)>
```

Wait for approximately 2 minutes for processes to terminate.

4. Enter another *ps* command to verify that all *Frame Exchange* and *Exchange Controller* processes have terminated.

Install

To install a CDS CD-1.1:

1. Rename the root of the CDS CD-1.1 install environment; for example,

```
mv <root-path>/cds11 <root-path>/cds11_old.
```
2. Move the tar to the root directory of the CDS CD-1.1 installation, and untar the CDS CD-1.1 delivery.
3. Verify that entries exist in the files */etc/inetd.conf* and */etc/services* to support automated startup of the CDS CD-1.1 soft-

▼ Installation Procedures

ware on applicable hosts. (See the [“Internet Daemon Configuration” on page 107.](#))

- On the *Connection Manager* host check `inetd.conf` for the following line:

```
connmgr stream tcp nowait <name> <root-path>/cds11/
sbin/run_idc_connmgr
```

- On the *Connection Manager* host check `services` for the following line:

```
connmgr 8031/tcp # CDS, CD-1.1 connection manager
```

- On all *Connection Manager Server* hosts check `inetd.conf` for the following line:

```
connsvr stream tcp nowait <name> <root-path>/cds11/
sbin/run_idc_connsvr
```

- On all *Connection Manager Server* hosts check `services` for the following line:

```
connsvr 8032/tcp # CDS, CD-1.1 connection server
```

4. Add required entries as necessary, and restart the Internet daemon.

5. Update files in the system specification area:

```
cd <root-path>/config/system_specs
```

6. Update the parameters in the `cds.par` file:

```
vi cds.par
```

Alter the definition of the following variables:

```
IDC<1-n>name
```

```
IDC<1-n>addr
```

```
NDC<1-n>name
```

7. Update the database account and the password definition in the file `process.par`:

```
vi process.par
```

```
IDCXDB=<user-name>/<passwr>@<DB-instance>
```

8. Update *CMS_HOME* in the global environment file:

```
cd env
vi global.env
setenv CMS_HOME <root-path>
```

9. Update the scripts in the bin directory:

```
cd <root-path>/bin
```

10. Update parameters in the *cds_idc_env* file:

```
vi cds_idc_env
```

Alter the definition of the following variable:

CMSroot (This should be set to the location of the *<root-path>*.)

11. Update the location of the environment script in run scripts:

```
vi run_idc_connmgr
```

Update the source line:

```
source <root-path>/bin/cds_idc_env
vi run_idc_connsvr
```

Update the source line:

```
source <root-path>/bin/cds_idc_env
```

When multiple *run_idc_connsvr* scripts exist, each one will need to be updated as above.

12. Create/rename *ForeMan* par files for each host running a *Data Center Manager*:

```
cd <root-path>/config/app_connfig/continuous_data/ ForeMan
mv ForeMan_idc_<old-name>.par ForeMan_idc_<new-host-name>.par
```

If the execution environment has changed (for example, because of new stations, new forwarding destinations, or a new installation root) then run the setup environment script.

For new station(s)/forwarding destination(s):

1. If the Frame Store definition has not been updated, provide the necessary updates. (See ["Frame Store Changes" on page 141](#)).

▼ Installation Procedures

2. If the `setup_idc_env` script has not been updated, provide the necessary updates. (See ["Environment Modification" on page 151.](#))
3. For any of the environment changes:
`cd <root-path>/utils/env`
4. Execute setup script for site:
`setup_env idc`

Restart CDS CD-1.1

To restart *CDS CD-1.1*:

1. Reinstate the *Connection Manager start* script:
`cd <root-path>/bin`
`mv run_idc_connmgr.sav run_idc_connmgr`
2. Start a *Data Center Manager (ForeMan)* process on each host servicing an outbound connection or a *Data Parser*:
`rlogin <host> -l <cds-name>`
`cd <root-path>/bin`
`run_idc_dcmgr <host> &`

ADDING DATA PROVIDERS

This section provides guidelines for adding a data provider to the operational environment at the IDC. The data provider may be an IMS station or a data center forwarding data to the IDC. In the information that follows, it is assumed that software is already installed and that all configuration files (except where explicitly stated otherwise) already exist and are functional.

The instructions that follow are presented in the order in which they shall be carried out. A fictitious station named KCC will be added; that station will supply data from a single site delivering three channels of data. The name for the data provider site has importance in the configuration of the software, due to the use of parameter substitution. As a consequence, ensure that the definition and use of the name in the configuration is unique and consistently applied, including the use of upper and lower case.

Frame Store Changes

Changes to the Frame Store must be coordinated with the supplier and receiver(s) of the *CDS CD-1.1* data, because the names of the frame sets must be consistent between providers and consumers of frames. For example, if the fictitious site KCC has one frame set for storing Data Frames named KCC:0, then the IDC must also use the name KCC:0 for storing these frames. Furthermore, forwarding destinations must also have a frame set KCC:0 for receiving forwarded data.

1. Update `<CONTINUOUS_DATA-DIR>/FrameStore fstore_idc.par` to support the new station. Add frame set entries to support the main flow of data with the frame set named KCC:0:

```
KCC:0          <index>      28800 864000 22000 0.25 3200 28800 86400 256
```

Comments in the par file describe each field on this data line and will not be repeated here. However, the `<index>` token would be replaced with a number identifying a unique index for entries in the Frame Store. No assumptions are made by the software about numbering schemes for indices, and human understanding should guide the selection and assignment.

2. Add two more frame sets to support "command" frames to and from the station. The indices for these two entries must also be unique as follows:

```
KCC:IDC        <index+1>    28800 864000   800 0.25 3200 28800 86400 256
IDC:KCC        <index+2>    28800 864000   800 0.25 3200 28800 86400 256
```

3. Add a frame set with a unique index to support *Exchange Controller* processing for the connection. Typically, the *Exchange Controller* frame sets will be grouped together:

```
CTLR:IDC-KCC   <index+200>    0         0         0 0.25 3200 28800 86400 256
```

4. Add frame sets needed by the *Data Parser* with unique indices:

```
KCC:DLPARSE    <index+1001>    0         0         0 0         0 20000   0
KCC:AUTH       <index+1002>    0         0         0 0         0 20000   0
```

▼ Installation Procedures

5. Insert frame set definitions following the par file table entries described in steps 1 through 4. Index values specified in these entries are the key fields that enable correlation of data definitions:

```
Channels[<index>]="KCC/shz KCC/shn KCC/she"
FrameLog[<index>]="$ (RUNroot)/FrameStore/KCC.flog"
Channels[<index+1>]="XX/sz XX/se XX/sn"
FrameLog[<index+1>]="$ (RUNroot)/FrameStore/kcc_idc.flog"
Channels[<index+2>]="XX/sz XX/se XX/sn"
FrameLog[<index+2>]="$ (RUNroot)/FrameStore/idc_kcc.flog"
Channels[<index+200>]="XX/sz XX/se XX/sn"
FrameLog[<index+200>]="$ (RUNroot)/FrameStore/ctrlr_idc-KCC.flog"
Channels[<index+1001>]=$(Channels[<index>])
FrameLog[<index+1001>]=$(RUNroot)/FrameStore/dlp_kcc_parse.flog
Channels[<index+1002>]=$(Channels[<index>])
FrameLog[<index+1002>]=$(RUNroot)/FrameStore/dlp_kcc_auth.flog
```

6. Insert definitions in the file for frame logging aspects of the new frame sets. Comments are provided in the par file relating to the meaning of the data fields and are not repeated here. Add the following:

```
#!/BeginTable KCC:0
| VolumeId | Dir | MaxSize
| 0 | $(RUNroot)/FrameStore | 1900800
#!/EndTable
#
#!/BeginTable KCC:IDC
| VolumeId | Dir | MaxSize
| 1 | $(RUNroot)/FrameStore | 69120
#!/EndTable
#
#!/BeginTable IDC:KCC
| VolumeId | Dir | MaxSize
| 1 | $(RUNroot)/FrameStore | 69120
#!/EndTable
#!/BeginTable CTRLR:IDC-KCC
| VolumeId | Dir | MaxSize
| 1 | $(RUNroot)/FrameStore | 69120
#!/EndTable
#
```

Connection Manager Changes

The `<CONTINUOUS_DATA-DIR>/ConnMgr` directory contains the par files used by the *Connection Manager* and the *Connection Manager Server*. When hosting connections on multiple machines, the following par files used by the *Connection Manager Server* will be unique for each host: `connsvr_<inst>.par`, `connsvr_<inst>_auth.par`, and `port_<inst>.par`. After selecting the computer that will service the inbound data flow, update the `port_<inst>.par` corresponding to the selection. Add an entry in this table to identify the new site and the communication port:

`port-KCC=<port-number>`

The value for `<port-number>` should be a unique value for the host computer: No other site or application program should use the identified port. Additionally, port-number assignment should be coordinated with the system administrator to ensure consistency in data center configuration.

Consumer Frame Exchange/Exchange Controller Setup

The *Frame Exchange* and *Exchange Controller* use the same par file, `<CONTINUOUS_DATA-DIR>/ExCtrl ctrl_fex_idc_sta.par`. Changes required in this file depend on how data are being delivered to the IDC by the provider. In the example being illustrated, no changes are required to support the new connection. The *Frame Exchange* and *Exchange Controller* will be correctly configured through the use of cascading parameter definitions and parameter substitution.

If the data provider has defined more than one frame set for the data flow, then simple variable substitution will not suffice. In this situation a unique .par is needed to support the connection:

1. Set `ctrl-reading-frame-set-list` in the new par file to the list of frame sets being provided by the provider. An example definition of this form is as follows:

▼ Installation Procedures

```
ctrlr-reading-frame-set-list="KCCn:0 KCCe:0 KCCz:0"
```

Forwarding Frame Exchange/Exchange Controller Setup

Forwarding instantiations of the *Frame Exchange* and *Exchange Controller* also share a configuration file and may be found in the same directory as the data consumer instantiation. Because each forwarding destination has a unique set of forwarded data, unique par configuration files are needed for each forwarding instance. Configuration is provided in a file named `ctrlr_fex_idc_ <dest>.par`, where `<dest>` identifies the destination consumer. To update such a file for a new station:

1. Update the parameter *ctrlr-polling-frame-set-list* to include the new frame sets for the station. Note that this will be the frame set written to by the *Frame Exchange* handling the inbound connection for the site. For example:

```
ctrlr-polling-frame-set-list="CD01:0 CD02:0 KCC:0"
```

2. Update the parameter *fex-reading-frame-set-list* to include the new site's frame set. For example:

```
fex-reading-frame-set-names="IDC:$(connection),  
CD01:0,CD02:0,KCC:0"
```

Authentication Updates

Authentication environment changes are necessary to include public key information of new sites. Public keys are maintained in the directory `<CONTINUOUS_DATA-DIR>/auth/sensor_pub`. This location may need to change if a different key management infrastructure is implemented. Obtain the public key file corresponding to the private key used by the site to sign frames and place it in this directory. Update the file `index.txt` in this directory to identify the new public key. Add an entry in the index file, for example:

```
KCC.pem|KCC|1|<key-id>|2000/04/28 00:00:00|2001/04/28  
00:00:00|
```

The fields in this line are separated by vertical bars ('|') and are defined from left to right as follows:

- filename: name of the public key file
- entity name: name of station/location
- active indicator: integer flag where 1 means key information for entity name is valid and active (authentication processing will take place, 0 means key information is inactive)
- key-identifier: integer index of key. This identifier used to identify the key corresponding to a signature and is provided in all protocol frames and Channel Subframes (this value must agree with the value used by the creator of a signature).
- not before: date and time of the beginning of a key's validity
- not after: data and time of the end of a key's validity

Database Definitions

Database updates are required to support connections from data providers and to process data received from providers. Definitions supporting processing in this context are limited to those provided by the software of the Continuous Data Subsystem. Other definitions are required to support processing in the larger monitoring system.

Refer to [\[IDC5.1.1Rev2\]](#) for documentation describing in detail the contents of the database and all of its tables.

As a method for carrying out required updates, create scripts files to affect database changes. This approach provides a record of what changes were introduced to the database and a template to use the next time a station needs to be added to the environment. The following sections present examples as if they were contained in a script file being read by a SQL*Plus interactive session.

▼ Installation Procedures

Affiliation Table

This table provides definitions that outline reporting sites and the association between the reporting array and the array's sites. An entry is required in this table even when a site is not part of an array; in this case the site's name would be provided as both the array/network-entity and the station entity. Update this table as follows:

```
REM*****
REM Insert into affiliation table
REM
insert into affiliation (net, sta, lddate) values ('KCC', 'KCC', sysdate);
```

If a site has decided to provide separate frame sets for each of its data sources, a separate entry would be required for each source, as well as entries to establish the relationship between the sources and the site. For example:

```
REM*****
REM Insert into affiliation table
REM
insert into affiliation (net, sta, lddate) values ('KCC', 'KCCn', sysdate);
insert into affiliation (net, sta, lddate) values ('KCC', 'KCCe', sysdate);
insert into affiliation (net, sta, lddate) values ('KCC', 'KCCz', sysdate);
insert into affiliation (net, sta, lddate) values ('KCCn', 'KCCn', sysdate);
insert into affiliation (net, sta, lddate) values ('KCCe', 'KCCe', sysdate);
insert into affiliation (net, sta, lddate) values ('KCCz', 'KCCz', sysdate);
```

Alphasite Table

The **alphasite** table identifies the allowable data providers to the data center and is used to validate connection requests. Update this table to add the new site/data provider as follows:

```
REM*****
REM Insert new station into alphasite table
REM
insert into alphasite (sta, address, prefdlid, time) values ('KCC', <ip-
<addr>, <dlid>, 0);
```

The `<ip-addr>` token is replaced with the IP address of the station/site in dot notation and enclosed in single quotes, for example, '123.456.78.9.' The `<dliid>` token is replaced with the integer identifier of the entry in the **dlman** table specifying which host/CDS-instance will service connections from this site.

DLman Table

Typically adding a site will not necessitate a modification to this table, but will require that the **alphasite** table contain a reference via the *dliid* field to an entry in this table.

The **dlman** table defines instances of *CDS CD-1.1* software. If the data provider connection hosting is distributed on the LAN at the IDC, this table will list available instances.

Sensor Table

The **sensor** database table provides definitions that describe the sensor at the site, for example:

```
REM *****
REM  Sensors for the new station/array
REM *****
insert into sensor (sta, chan, time, endtime, inid, chanid,
                  jdate, calratio, calper, tshift, instant,
                  lddate)
values
('KCC', 'she', 946684800.000, 999999999.999, 50000,
 <chanid>, 2000001, <calratio>, <calper>, 0, 'y',
 sysdate); insert into sensor (sta, chan, time, endtime,
 inid, chanid, jdate, calratio, calper, tshift, instant,
 lddate) values ('KCC', 'shn', 946684800.000,
 999999999.999, 50000, <chanid>, 2000001, <calratio>,
 <calper>, 0, 'y', sysdate);
insert into sensor (sta, chan, time, endtime, inid, chanid,
                  jdate, calratio, calper, tshift, instant,
                  lddate)
```

▼ Installation Procedures

```

values
('KCC', 'shz', 946684800.000, 999999999.999, 50000,
 <chanid>, 2000001, <calratio>, <calper>, 0, 'y',
 sysdate);

```

Site Table

The **site** database table provides information describing the site. Most of this information is not used by the *CDS CD-1.1* software. This table has entries for the arrays and for each site in an array. Add an entry similar to the following:

```

REM *****
REM Insert into Site table
REM This table is used mostly for analysis purposes;
REM First insert is for the reference station when inserting an array
REM Following entries are for array's elements
REM *****

insert into site (sta, ondate, offdate, lat, lon, elev, staname,
                 statype, refsta, dnorth, deast, lddate)
values
('KCC', 2000001, -1, <array-ref-lat>, <array-ref-lon>,
 <array-ref_elev>, 'New Station KCC', 'ar', 'KCC', 0.000,
 0.000, sysdate);
insert into site (sta, ondate, offdate, lat, lon, elev, staname,
                 statype, refsta, dnorth, deast, lddate)
values
('KCC', 2000001, -1, <sensor-lat>, <sensor-lon>,
 <sensor_elev>, 'New Station KCC', 'ss', 'KCC', 0.000,
 0.000, sysdate);

```

Sitechan Table

The **sitechan** database table describes the channels of data coming from a site. Add definitions similar to the following:

```

REM *****
REM Insert into SiteChan table
REM
REM *****
REM sta KCC chans she, shn, shz
REM *****

```

```

insert into sitechan (sta, chan, ondate, chanid, offdate, ctype,
                     edepth, hang, vang, descrip, lddate)
values
    ('KCC', 'she', 2000001, <chan-id>, -1, 'n', .01, 90.0,
     90.0, 'short-period e-w', sysdate);
insert into sitechan (sta, chan, ondate, chanid, offdate, ctype,
                     edepth, hang, vang, descrip, lddate)
values
    ('KCC', 'shn', 2000001, <chan-id>, -1, 'n', .01, 90.0, 0.0,
     'short-period n-s', sysdate);
insert into sitechan (sta, chan, ondate, chanid, offdate, ctype,
                     depth, hang, vang, descrip, lddate)
values
    ('KCC', 'shz', 2000001, <chan-id>, -1, 'n', .01, 0.0, 0.0,
     'short-period vertical', sysdate);

```

Wfconv Table

The **wfconv** database table provides data that are used to convert data into the CSS 3.0 format, which is used by both the analysis and display processes. Add entries for each channel of data being provided as follows:

```

REM *****
REM sta KCC chans she, shn, shz
REM *****
REM
insert into wfconv (sta, chan, chanid, inauth, incomp, intype, insamp,
                  outauth, outcomp, outtype, outsamp, strip,
                  commid, lddate)
values
    ('KCC', 'she', <chan-id>, 'y', <comp-type>, <in-type>, 400,
     'n', '-', <out-type>, 400, 'y', -1, sysdate);
insert into wfconv (sta, chan, chanid, inauth, incomp, intype, insamp,
                  outauth, outcomp, outtype, outsamp, strip,
                  commid, lddate)
values
    ('KCC', 'shn', <chan-id>, 'y', <comp-type>, <intype>, 400,
     'n', '-', <out-type>, 400, 'y', -1, sysdate);
insert into wfconv (sta, chan, chanid, inauth, incomp, intype, insamp,
                  outauth, outcomp, outtype, outsamp, strip,
                  commid, lddate)

```

▼ Installation Procedures

```

values
('KCC', 'shz', <chan-id>, 'y', <comp-type>, <in-type>, 400,
'n', '-', <out-type>, 400, 'y', -1, sysdate);

```

In the above commands:

- The *<comp-type>* token is either the single-quoted string 'ca' designating Canadian Compression or '-' designating no data compression.
- The *<in-type>* token is one of the data types specified in the *CDS CD-1.1 Formats and Protocols* [\[IDC3.4.2\]](#) document, enclosed in single quotes, for example, 's4,' 's3,' and 's2'.
- The *<out-type>* token is one of the data types specified in the *CDS CD-1.1 Formats and Protocols* [\[IDC3.4.2\]](#) document, enclosed in single quotes.

Wfproto Table

The **wfproto** database table provides prototype information for the creation of waveform description records (**wfdisc**) for time-series data. Add definitions similar to the following:

```

REM *****
REM sta KCC chans she, shn, shz
REM *****
insert into wfproto (sta, chan, time, wfid, chanid, jdate, endtime,
                    nsamp, samprate, calib, calper, instype,
                    segtype, datatype, clip, dir, dfile, foff,
                    commid, lddate)
values
('KCC', 'she', 946684800.000, -1, <chan-id>, 2000001,
999999999.999, 0, <samp-rate>, <calib>, <calper>,
'NOTYPE', '-', <in-type>, '-', '-', '-', 0, -1, sysdate);
insert into wfproto (sta, chan, time, wfid, chanid, jdate, endtime,
                    nsamp, samprate, calib, calper, instype,
                    segtype, datatype, clip, dir, dfile, foff,
                    commid, lddate)
values
('KCC', 'shn', 946684800.000, -1, <chan-id>, 2000001,
999999999.999, 0, <samp-rate>, <calib>, <calper>,
'NOTYPE', '-', <in-type>, '-', '-', '-', 0, -1, sysdate);

```

```

insert into wfproto (sta, chan, time, wfid, chanid, jdate, endtime,
                    nsamp, samprate, calib, calper, instype,
                    segtype, datatype, clip, dir, dfile, foff,
                    commid, lddate)
values
('KCC', 'shz', 946684800.000, -1, <chan-id>, 2000001,
 999999999.999, 0, <samp-rate>, <calib>, <calper>,
'NOTYPE', '-', <in-type>, '-', '-', '-', 0, -1, sysdate);

```

Environment Modification

The directory <CMSroot>/utils/env contains scripts that aid in creating and modifying an environment for the CDS CD-1.1 software. These scripts operate with the Frame Store and the database definitions provided in the previous paragraphs. Make the following change in the file <CMSroot>/utils/env/setup_idc_env.

1. Update the definition of *SET_LIST* that identifies the list of frame sets supporting the flow of data from data providers. For example:

```

setenv SET_LIST "CD01:0 CD01:IDC IDC:CD01 CD02:0
CD02:IDC IDC:CD02 KCC:0
KCC:IDC IDC:KCC"

```

This definition must reflect the definitions in the Frame Store. If the data provider has defined different frames set allocations, they must be represented in this list. This list will be used to create the frame sets.

The assignment of this environment variable is repeated later for the purpose of identifying frame sets required by the *Data Parser*.

2. Add the new station to this later instance, similar to the following:

```

setenv SET_LIST "CD01:DLPARSE CD01:AUTH CD02:DLPARSE
CD02:AUTH KCC:DLPARSE KCC:AUTH"

```

The same caveat applies to this list as to that specified for Data Frame sets.

▼ Installation Procedures

3. The *SET_LIST* variable is redefined later in the file to identify the special logging frame sets used by the *Exchange Controller*. Add an entry to the list specifying this frame set. For example:

```
setenv SET_LIST "CTLR:IDC-CD01 CTLR:IDC-CD02 CTLR:IDC-KCC"
```

4. Later in the file the *SET_LIST* variable is used to identify the Disk Loops to be created. Update this list to add needed Disk Loops similar to the following:

```
setenv SET_LIST "CD01 CD02 KCC"
```

This list is also influenced by the way frame sets are defined by the provider.

5. Execute the script setup with the *idc* command and modifier as follows:

```
setup_env idc
```

This script creates Disk Loop files, frame log files, and the frame set directories needed to support the new station. After this job is completed the data center is prepared to support connections from the new site.

SHORT COURSE FOR ADDING DATA PROVIDERS

This section provides an abbreviated list of activities for adding a data provider to the data center environment. The activities are terse and assumes that you are familiar with the configuration and operation of the *CDS CD-1.1* software. For more detailed instructions see ["Adding Data Providers" on page 140](#). All activities assume a configuration consistent with instructions in this document.

Frame Store Changes

```
cd <root-path>/config/app_config/continuous_data/FrameStore
```

1. Add frame sets, for example:

```
CD01:0  
CD01:IDC  
IDC:CD01  
CTLR:IDC-CD01  
CD01:DLPARSE  
CD01:AUTH
```

2. Add the necessary frame log entries for the new frame sets.

Connection Manager Changes

```
cd <root-path>/config/app_config/continuous_data/ConnMgr
```

1. Add the data provider to the port_<inst>.par file.

Forwarding Frame Exchange/Exchange Controller Setup

```
cd <root-path>/config/app_config/continuous_data/ExCtrlr
```

A change is needed only if the forwarding location is to receive data from the new data provider being added:

1. Update the .par files of those forwarding (*Frame Exchange/Exchange Controller*) instantiations that are to receive the new data provider's data.
2. Update the ctrl-polling-frame-set-list and fex-reading-frame-set-names to include the new frame set, for example, CD01:0.

▼ Installation Procedures

Data Parser Changes

```
cd <root-path>/config/app_config/continuous_data/DLParse
```

Update .par file for an instance of the *Data Parser* that will parse the new data provider's data to include the provider's name:

1. Update `sta_fs_list` with the provider/frame set pair. For example:
`sta_fs_list="CD01 CD01:0"`
2. Add a *last id* token for the provider:
- `CD01_id=1`
3. Add a size token for the provider:
- `CD01_sz=40000`

Authentication Updates

```
cd <root-path>/config/app_config/continuous_data/auth
```

1. Install/copy the station's public key(s) into the public key directory:
`<root-path>/config/app-config/continuous_data/auth/
sensor_pub`
2. Update the `index.txt` file in the public key directory to identify the new key.

Database Definitions

The following tables are updated for use by the *CDS CD-1.1*: **affiliation**, **alphasite**, **dlman**, **sensor**, **site**, **sitechan**, **wfconv** and **wfproto**. Other monitoring system applications require updates to additional tables to support analysis of data from a new sources; those tables are not addressed here.

Environment Modifications

```
cd <root-path>/utils/env
```

1. Add new frame sets to the `SET_LIST` definitions in the `setup_idc_env` file.

2. Add a station name to the *SET_LIST* definitions in the *setup_idc_env* file for identifying Disk Loops to be created.
3. Rerun the environment setup script:

```
setup_env idc
```

ADDING FORWARDING DESTINATION

This section provides a guideline for adding a data consumer (forwarding location) to the operational environment at the IDC. The information assumes that the software is already installed and that all configuration files (except where explicitly stated otherwise) already exist and are functional.

Instructions are presented in the order in which they are recommended to be carried out. A fictitious location named NDC1 will be added to the operational environment. The name for the data consumer site has importance in the configuration of the software because of the employment of variable substitution. Ensure that the definition is unique and consistently applied, including the use of upper and lower case.

Subsystem-level Changes

The *cds.par* file contains high-level subsystem parameter definitions, including the identification of the forwarding destinations. This file can be found at *<CMSroot>/config/system_specs*.

1. Update this file to add the name of the host computer to be contacted at the new destination. This would be the machine hosting the *Connection Manager* process at the remote location. Add the definition:

```
NDC1name=<forward-dest>
```

▼ Installation Procedures

The token *<forward-dest>* can be an alphanumeric name, for example, `amazon.cmr.gov`, or a dot notation IP address, such as `123.45.678.9`. However, alphanumeric names are preferred if used; the name must be present in the `/etc/host` file of the computer (or be available through the Network Information System [NIS] as applicable).

Frame Store Changes

Changes to the Frame Store must be coordinated with the supplier(s) and receiver(s) of the *CDS CD-1.1* data, because the names of the frame sets must be consistent between providers and consumers of frames. See the discussion for Frame Store changes under [“Adding Data Providers” on page 140](#), for additional information.

1. In the directory *<CONTINUOUS_DATA-DIR>/FrameStore*, update the file `fstore_idc.par` to add the forwarding destination. Add frame set entries to support the command frame sets needed for the connection, that is, one from the IDC to the new destination and one from the new destination to the IDC as follows:

```
IDC:NDC1      <index>      28800 864000   800 0.25 3200 28800 86400 256
NDC1:IDC      <index+1>    28800 864000   800 0.25 3200 28800 86400 256
```

The `par` file contains comments that describe what each of the fields are on this data line; they will not be repeated here. However, the *<index>* token would be replaced with a number identifying a unique index for entries in the Frame Store. No assumptions are made by the software about numbering schemes for indices; human knowledge should guide selection and assignment.

2. A frame set is needed to support *Exchange Controller* processing for the connection. Add an entry with a unique index:

```
CTLR:IDC-NDC1 <index+200>    0      0      0 0.25 3200 28800 86400 256
```

3. Follow the par file table entries described above with further definitions needed for the frame sets. Index values specified in these entries are the key fields enabling the correlation of data definitions. Channel information is provided in these entries for completeness and is not mapped to any real data. Add the following definitions:

```
Channels[<index>]="XX/sz XX/se XX/sn"
Framelog[<index>]="$ (RUNroot)/FrameStore/idc_NDC1.flog"
Channels[<index+1>]="XX/sz XX/se XX/sn"
Framelog[<index+1>]="$ (RUNroot)/FrameStore/NDC1_idc.flog"
Channels[<index+200>]="XX/sz XX/se XX/sn"
Framelog[<index+200>]="$ (RUNroot)/FrameStore/ctrlr_idc-NDC1.flog"
```

4. Provide definitions further in the file for the frame logging aspects of the new frame sets. Comments are provided in the par file relating the meaning of data fields and are not repeated here. Add the following definitions:

```
#!BeginTable IDC:NDC1
| VolumeId | Dir | MaxSize
| 1 | $(RUNroot)/FrameStore | 69120
#!EndTable
#
#!BeginTable NDC1:IDC
| VolumeId | Dir | MaxSize
| 1 | $(RUNroot)/FrameStore | 69120
#!EndTable
#!BeginTable CTRLR:IDC-NDC1
| VolumeId | Dir | MaxSize
| 1 | $(RUNroot)/FrameStore | 69120
#!EndTable
#
```

Connection Originator Setup

The *Connection Originator's* configuration file may be found at <CONTINUOUS_DATA-DIR>/ConnOrig with the name ConnOrig_<inst>.par, where <inst> identifies the forwarding connection.

▼ Installation Procedures

Update the definition of the contact port at the remote site:

```
co-connection-manager-ports="$($ (connection)name):  
<port-num>"
```

The token *<port-num>* is a configurable parameter by the forwarding destination and must be coordinated with the system administrator for the remote machine. Depending on how the *Data Center Manager's* configuration has been defined, additional changes may be needed. In particular, in the above definition the *\$(\$ (connection)name)* string substitutes variables as follows: *\$ (connection)* is replaced with a destination (NDC1), which is expected to be provided by the *Data Center Manager's* par file. The resulting variable *\$ (NDC1name)* is replaced with the machine name or IP address from the *cds.par* configuration file. If these substitutions are not available or are not desired, a suitable definition would be provided to specify the name.

If the *Data Center Manager's* configuration does not provide definitions for the variables *self* and *connection*, then the following definitions must be updated to provide the correct name of the *Exchange Controller's* par file and the name of the local location (IDC), respectively:

```
co-exch-ctlr-pars="par=$(CONTINUOUS_DATA-DIR)/ExCtlr/  
ctlr_fex_$(self)_$(connection).par"  
co-station-name=$(self)
```

Frame Exchange/Exchange Controller Setup

The *Frame Exchange* and *Exchange Controller* share a configuration file in the directory *<CONTINUOUS_DATA-DIR>/ExCtlr*. Because each forwarding destination most likely will have a unique set of forwarded data, unique par configuration files are likely to be needed for each forwarding instantiation. Configuration is provided in a file named *ctlr_fex_IDC_<dest>.par*, where *<dest>* identifies the destination consumer. To support a new destination an existing forwarding instance par file should be copied to a new name. Similarly an existing version of the file provid-

ing logging parameters can be copied. Changes then are made to the new files to establish needed definitions. To create new configuration files issue a command similar to the following:

```
cp ctrl_fex_IDC_ABC.par ctrl_fex_IDC_NDC1.par
cp ctrl_errlog_idc_ABC.par ctrl_errlog_idc_NDC1.par
```

In the new file `ctrl_fex_IDC_NDC1.par`:

1. Update the line identifying the error logging par file to contain the newly copied file. Enter a command similar to the following:

```
par="$(CONTINUOUS_DATA-DIR)/ExCtrl/
ctrl_errlog_idc_NDC1.par"
```

2. Provide the list of frame sets to be forwarded to the new destination. For example:

```
ctrl-polling-frame-set-list="CD01:0 CD02:0 KCC:0"
```

3. Update the line identifying the error logging par file to be used by the *Frame Exchange* process. Enter a command similar to the following:

```
fex-logpar-file="$(CONTINUOUS_DATA-DIR)/FrameEx/
fex_errlog_idc_NDC1.par"
```

In the new file `ctrl_errlog_idc_NDC1.par`:

1. Verify that the parameter giving the name of the log file is either defined using variable substitution or contains a unique name. A definition similar to the following should exist:

```
log-name=ctrl_IDC_$(connection)
```

2. Change to the *Frame Exchange* configuration directory, `<CONTINUOUS_DATA-DIR>/FrameEx`, and create a new error logging configuration file by copying an existing instance with a command similar to the following:

```
cp fex_errlog_idc_ABC.par fex_errlog_idc_NDC1.par
```

The new file must have the same name as provided in the combined *Exchange Controller* and *Frame Exchange* par file for defining the parameter `fex-logpar-file`.

▼ Installation Procedures

In the new file `fex_errlog_idc_NDC1.par`:

1. Verify that the parameter giving the name of the log file is either defined using variable substitution or contains a unique name. A definition similar to the following should exist:

```
log-name=fex_IDC_$(connection)
```

Data Center Manager Setup

A *Data Center Manager* instance must be updated to start and manage the processes to service a new connection.

1. Select a computer to host the *Data Center Manager*. The host machine must have connectivity to the WAN and visibility to the disk partition where the Frame Store is defined.
2. Update the run-time configuration in the file `<CONTINUOUS_DATA_DIR>/ForeMan/ForeMan_idc_<inst>.par`. The token `<inst>` should identify the host computer. For example, a *Data Center Manager* running on the machine `amazon` would result in the configuration file `ForeMan_idc_amazon.par`.
3. If there is not an existing file for this instance of the *Data Center Manager* copy an existing *ForeMan-*.par* file.
4. Add support for the new forwarding destination with the following modifications to the *Data Center Manager's* `par` file.

Update the parameter that defines the name of the log file. The name must be unique among all *Data Center Manager* instances to avoid logging clashes. The name should include the host name to provide uniqueness and understandability. For example:

```
log-name=foreman_IDC_amazon
```

Next add a series of definitions to identify and create a job template. The *Data Center Manager* uses the templates to identify and control processes.

In the following examples a new template is defined for a *Connection Originator* instance connecting to the NDC1 forwarding destination.

1. Add the identifier of a new job to the template identification list (which originally just listed the *Data Parser* job):

```
fm-job-template-ids='DLParse ConnOrig-NDC1'
```

2. Copy one of the job templates in the configuration file to accommodate the definitions needed for the new job. The following example is an entire template with the needed definitions provided:

```
#####
# Begin template
#
# Where process is located
fm-ConnOrig-NDC1-exec-path='${BINroot}/ConnOrig self=IDC
connection=NDC1 par=${CONTINUOUS_DATA-DIR}/ConOrig/ConnOrig_
NDC1.par'
# ForeMan judges a successful startup based on the number of
# seconds the process runs (before exiting). This item defines
# the minimum amount of time process needs to run.
fm-ConnOrig-NDC1-min-runtime=600 # 10 minutes
# How many times to restart job, after an un-natural termination,
# i.e., runs less than fm-<name>-min-runtime seconds.
fm-ConnOrig-NDC1-max-restarts=6
# How long is seconds between restarts, when an un-natural
# termination
# occurs. NOTE, the number of values in this list is equal to
# the value of fm-<name>-max-restarts
fm-ConnOrig-NDC1-restart-waits=10.0,30.0,60.0,90.0,300.0,900.0
# Command to run (what to do) after fm-<name>-max-restarts is
# exceeded
# Note that if this item identifies itself, ForeMan will
# continually
# attempt to restart/run this job.
fm-ConnOrig-NDC1-command-on-failure='jobreq 1.0 ConnOrig <<>>'
# Indicator for whether job will provide a ForeMan command (read
# from a socket connection) - (command must be exactly in the
# format of fm-<name>-command-on-failure), 1 -> yes, 0 -> no
fm-ConnOrig-NDC1-command-and-control=0
# End template
#####
```

▼ Installation Procedures

3. After the template definition updates, modify the list of initial events to process to include the new jobs, as follows:

```
fm-initial-events='\
jobreq 10.0 DLParse      <<>>  \
jobreq  0.0 ConnOrig-NDC1 <<>>  \
'
```

In this example the *Data Parser* job was an existing managed job.

Authentication Updates

Authentication environment changes are necessary to include public key information of the forwarding destination. Most frame traffic will be from the data center to the new forwarding destination. However, connection protocol frames will be received, and then the *CDS CD-1.1* will need the new destination's public key. Public keys are maintained in the directory `<CONTINUOUS_DATA-DIR>/auth/sensor_pub`. This location may need to change if a different key management infrastructure is implemented. Obtain the public key file corresponding to the private key used by the destination to sign frames, and place it in this directory. The file `index.txt`, which will need a new entry to identify the new public key, is in this directory.

1. Add an entry in the index file. For example:

```
NDC1.pem|NDC1|1|<key-id>|2000/04/28 00:00:00|2001/04/28 00:00:00|
```

The fields in this line are separated by the vertical bar (|). From left to right they are:

- filename—name of the public key file
- entity name—name of station/location
- active indicator—integer flag where 1 means key information for entity name is valid and active (authentication processing will take place); 0 means key information is inactive.

- key-identifier-integer index of key—This identifier used to identify the key corresponding to a signature and is provided in all protocol frames and Channel Subframes (this value must agree with the value used by the creator of a signature).
- not before—date and time of the beginning of a key's validity
- not after—data and time of the end of a key's validity

Environment Modification

The directory `<CMSroot>/utils/env` contains scripts that aid in creating and modifying an environment for *CDS CD-1.1* software. These scripts operate with the Frame Store provided in the above paragraphs. Make the following changes in the file `<CMSroot>/utils/env/setup_idc_env`:

1. Update the definition of `SET_LIST` used to identify the list of frame sets supporting connections to forwarding locations. For example, the NDC1 destination is added to an existing list.

```
setenv SET_LIST "IDC:ABC IDC:ABC IDC:NDC1 NDC1:IDC"
```

This definition must reflect the definitions in the Frame Store. This list will be used to create the frame sets.

2. The `SET_LIST` variable is redefined later in the file to identify the special logging frame sets used by the *Exchange Controller*. Add an entry to the list specifying this frame set. For example:

```
setenv SET_LIST "CTLR:IDC-CD01 CTLR:IDC CD02 CTLR:IDC-KCC  
CTLR:IDC-NDC1"
```

3. Execute this script as follows:

```
setup_env idc
```

This script will then create frame log files and frame set directories needed to support the new destination. After this job is completed, the data center is prepared to support connection to the new forwarding destination.

References

The following sources supplement or are referenced in document:

- [IDC3.4.2] Science Applications International Corporation, *Formats and Protocols for Continuous Data*, SAIC-98/3005, 1998.
- [IDC3.4.3] Science Applications International Corporation, *Formats and Protocols for Continuous Data CD-1.1*, SAIC-00/3026, 2000.
- [IDC5.1.1Rev2] Science Applications International Corporation, Veridian Pacific-Sierra Research, *Database Schema, Revision 2*, SAIC-00/3057, PSR-00/TN2830, 2000.
- [IDC7.4.1] Science Applications International Corporation, *Continuous Data Subsystem CD-1.1 Software User Manual*, SAIC-01/3012, 2001.

Appendix: Tools

This Appendix includes the following topics:

- [Environment Creation](#)
- [Frame Set Maker](#)
- [Make Loop \(Mkloop\)](#)

Appendix: Tools

The sections below provide examples of scripts that aid in the configuration of *CDS CD-1.1*. These scripts must be modified to reflect real data providers, data consumers, frame sets, and so on, of a data center.

ENVIRONMENT CREATION

The following is the top-level script for creation of a run-time environment. The script in turn will invoke lower-level scripts for definition of the environment.

```
#!/bin/csh
#####
# This script will setup a directory environment for testing the
# CD-1.1 CDS.
#
#####
#
if ($1 == "") then
    echo ""
    echo "You must provide indication of system being set-up"
    echo "Usage:  setup_env  <sys>"
    echo ""
    echo "          where: sys -> cd{1-10}, idc, or ndc"
    echo ""
    exit
endif

setenv SRCcloc `pwd`

# validate input argument with the first switch
#
switch ($1)
    case idc:
    case IDC:
        echo "Creating environment for IDC "
        Source cds_paths idc.env
```

```

        echo ""
        mkdir -p $LOGroot
        echo "Created cds log directory at " $LOGroot

        mkdir -p $RUNroot/FrameStore
        echo "Created frame store directory at " $RUNroot/
        FrameStore breaksw

    default:
        echo "Unrecognized system option, no action will be taken"
        exit -1;
endsw

# Now actually go create the environment
switch ($1)
    case idc:
        case IDC:
            source cds_paths_idc.env
            setup_idc_env
            breaksw

    default:
        echo "Unrecognized system option, no action will be taken"
        exit -1;
endsw

exit

```

The following is the script that is used to setup a run-time environment at the IDC :

```

#####
#
# Do NOT run this script directly from the command line.
#
# This script relies on some environment variables and the existence
# of directories which are provided by the setup_env script. It is
# intended that this script will be called from the setup_env script.
# Other uses will provide unacceptable results.
#
# This script will setup an execution environment needed for the CDS
# at a simulated IDC
#
#####
#
#
# This next section is dependent on the names of the frame sets used

```

▼ Tools

```

# by the CDS.
#
# See the Readme file in this directory for information about
# how to determine what the frame set names are.
#
#####

# for each frame set group
#   identify the (unique) frame sets in the group
#   create the frame set directories
#   create the frame sets
#
# First one
#
# Establish a list of the frame sets
#   For ease of viewing an maintenance this has been broken up into
#   multiple steps, they can easily be collapsed if necessary.

setenv SET_LIST "CD01:0 CD01:IDC IDC:CD01 CD02:0 CD02:IDC IDC:CD02
CD03:0 CD03:IDC IDC:CD03 CD04:0 CD04:IDC IDC:CD04 CD05:0 CD05:IDC
IDC:CD05 CD06:0 CD06:IDC IDC:CD06 CD07:0 CD07:IDC IDC:CD07 CD08:0
CD08:IDC IDC:CD08 CD09:0 CD09:IDC IDC:CD09 CD10:0 CD10:IDC IDC:CD10"

#
# Create directories for frame sets in the frame store
#
foreach i ($SET_LIST)
    mkdir $RUNroot/FrameStore/$i
    echo ""
    echo "Created frame set directory for xset " $i
    echo "          at " $RUNroot/FrameStore/$i
end

#
# Now run program for creating the frame sets, - it really just
# creates the frame logs that are needed by the frame store library
# software.
#

echo ""
echo "Setting up contents of frame store"

# Note that the fset_maker will use the par library to access the
# environment variable SET_LIST defined for this group

fset_maker par=fset_maker_idc.par

```

```
#####
# for each frame set group
#   identify the (unique) frame sets in the group
#   create the frame set directories
#   create the frame sets
#
# Next one
#
# Establish a list of the frame sets
#   These sets are for forwarding

setenv SET_LIST "IDC:NDC1 NDC1:IDC"

#
# Create directories for frame sets in the frame store
#
foreach i ($SET_LIST)
    mkdir   $RUNroot/FrameStore/$i
    echo    ""
    echo    "Created frame set directory for set " $i
    echo    "          at " $RUNroot/FrameStore/$i
end

#
# Now run program for creating the frame sets, - it really just
# creates the frame logs that are needed by the frame store library
# software.
#

echo ""
echo "Setting up contents of frame store"

# Note that the fset_maker will use the par library to access the
# environment variable SET_LIST defined for this group

fset_maker par=fset_maker_idc.par

#####
# for each frame set group
#   identify the (unique) frame sets in the group
#   create the frame set directories
#   create the frame sets
#
# Next one
```

▼ Tools

```

#
# Establish a list of the frame sets
# These are special logging sets for the exchange controller

setenv SET_LIST "CTLR:IDC-CD01 CTLR:IDC-CD02 CTLR:IDC-CD03 CTLR:IDC-
CD04 CTLR:IDC-CD05 CTLR:IDC-CD06 CTLR:IDC-CD07 CTLR:IDC-CD08
CTLR:IDC-CD09 CTLR:IDC-CD10 CTLR:IDC-NDC1"

#
# Create directories for frame sets in the frame store
#
foreach i ($SET_LIST)
    mkdir $RUNroot/FrameStore/$i
    echo ""
    echo "Created frame set directory for set " $i
    echo "      at " $RUNroot/FrameStore/$i
end

#
# Now run program for creating the frame sets, - it really just
# creates the frame logs that are needed by the frame store library
# software.
#

echo ""
echo "Setting up contents of frame store"

# Note that the fset_maker will use the par library to access the
# environment variable SET_LIST defined for this group

fset_maker par=fset_maker_idc.par

#####
# for each frame set group
#   identify the (unique) frame sets in the group
#   create the frame set directories
#   create the frame sets
#
# Next one
#
# Establish a list of the frame sets
# These are sets for the data parser

setenv SET_LIST "CD01 CD02 CD03 CD04 CD05 CD06 CD07 CD08 CD09 CD10"

```

```

#
# Create directories for frame sets in the frame store
#
foreach i ($SET_LIST)
    mkdir    $RUNroot/FrameStore/$i
    echo    ""
    echo    "Created frame set directory for set " $i
    echo    "          at " $RUNroot/FrameStore/$i
end

#
# Now run program for creating the frame sets, - it really just
# creates the frame logs that are needed by the frame store library
# software.
#

echo ""
echo "Setting up contents of frame store"

# Note that the fset_maker will use the par library to access the
# environment variable SET_LIST defined for this group

fset_maker par=fset_maker_idc.par

#####

```

FRAME SET MAKER

The program *fset_maker* exists for the purpose of establishing a frame store, frame set, and its accompanying frame log. *fset_maker* is executed from within the *setup_idc_env* script when creating a run-time environment. The processing in this program is rather simple and straight forward. It creates the frame store elements by issuing a request to open them with control flags that specify the creation of entities if they are not found. *fset_maker* inherits par file parameter definitions from the *setup_idc_env* script and requires no modification.

▼ Tools

MAKE LOOP (MKLOOP)

The program *Mkloop* establishes Disk Loop files for use by the *Data Parser*. The Disk Loops used by *CDS CD-1.1* software are the same as those used by the *CDS CD-1.0* software. Prior to creating Disk Loops, the database must be populated with information about the sensors and stations providing data. See the configuration description for the *Data Parser* (["Data Parser" on page 123](#)) for more information about required table definitions. *Mkloop* is executed from within the *setup_idc_env* script. However, it can also be executed as a stand-alone program.

Mkloop requires *libpar*-style command line strings for definition of run-time configuration values that result in the creation of Disk Loop files that are specific for a given reporting station. It is recommended that a *par* file be used when invoking *Mkloop* as follows:

```
Mkloop sta=<station-name> par=Mkloop.par
```

The *par* file *Mkloop.par* needs to be edited as identified above. The following items are those which would typically require modification.

- Set parameter *sta* to be the station where data originated.
- Define *machine* to be the dot notation IP address of the host where the *Data Parser* that uses this Disk Loop will execute.
- Set *samprate* to the sample rate (Hz) of data from station *sta*.
- Set *datatype* to the two character data type of data from station *sta*. See *CD-1.1 Formats and Protocols for Continuous Data* [\[IDC3.4.3\]](#) for supported data types.
- Set *dlid* to the integer ID of the *Data Parser* that will utilize this Disk Loop. See ["Data Parser" on page 123](#).

Glossary

A

array

Collection of sensors distributed over a finite area (usually in a cross or concentric pattern) and referred to as a single station.

ASCII

American Standard Code for Information Interchange. Standard, unformatted 256-character set of letters and numbers.

authentication signature

Series of bytes that are unique to a set of data and that are used to verify the authenticity of the data.

authenticate

Verify the authenticity of a string of bits with an authentication signature.

AutoDRM

Automatic Data Request Manager.

C

CD-ROM

Compact Disk–Read Only Memory.

channel

Component of motion or distinct stream of data.

child process

UNIX process created by the *fork* routine. The child process is a snapshot of the parent at the time it called *fork*.

connection

Open communication path between protocol peers.

CPU

Central Processing Unit.

CSC

Computer Software Component.

CSCI

Computer Software Configuration Item.

D

daemon

Executable program that runs continuously without operator intervention. Usually, the system starts daemons during initialization. (Example: cron.)

▼ Glossary

data center

Location receiving data from multiple data providers. Often this location will also process and forward time-series data received.

data consumer

Receiver of CD-1.1 data frames. This is always a data center.

data provider

Sender of CD-1.1 data frames. This may be a station or a data center that forwards data.

DBMS

Database Management System.

disk loop

Storage device that continuously stores new waveform data while simultaneously deleting the oldest data on the device.

F**firewall**

Software used to protect a computer or computer network from unauthorized access.

fork

UNIX system routine that is used by a parent process to create a child process.

frame

Logical collection of digital information that is transmitted as a unit from application to application.

frame set

Element of a frame store. Frame sets are defined for each source and destination pair. Each frame set has an associated frame log, which is also an element of the frame store.

frame store

Disk directories and files used to store raw CD-1.1 protocol frames. A frame store is made up of frame sets.

FTP

File Transfer Protocol; protocol for transferring files between computers.

G**GB**

Gigabyte. A measure of computer memory or disk space that is equal to 1,024 megabytes.

H**host**

Machine on a network that provides a service or information to other computers. Every networked computer has a hostname by which it is known on the network.

I**IDC**

International Data Centre.

IMS

International Monitoring System.

instance

Describes a running computer program. An individual program may have multiple instances on one or more host computers.

I/O

Input/Output.

IP

Internet protocol.

IP address

Internet Protocol address, for example: 140.162.1.27.

K**key**

Data string used by authentication software. Typically keys are defined in pairs, public and private. The private key is used to sign data (produce a validation data value), and the public key is used to verify data (determine that a validation data value was produced by the private counterpart of the public key).

L**LAN**

Local Area Network.

M**mutex**

Mutual exclusion lock used to prevent multiple threads from simultaneously executing critical sections of code which access shared data.

N**NDC**

National Data Center.

NFS

Network File System (Sun Microsystems). Protocol that enables clients to mount remote directories onto their own local filesystems.

NIS

Network Information System.

O**ORACLE**

Vendor of PIDC and IDC database management system.

P**parameter (par) file**

ASCII file containing values for parameters of a program. Par files are used to replace command line arguments. The files are formatted as a list of `[token = value]` strings.

▼ Glossary

parse

Decompose information contained in a set of data.

PID

Process Identifier.

PIDC

Prototype International Data Centre.

pipe

Interprocess communication facility provided by the UNIX operating system. Pipes typically are defined in pairs to support data transmission between two processes where each pipe supports a one-way flow of data.

port

Connection to a computer.

protocol peer

Computer system participating in an exchange of data using a specific protocol (for example CD-1.1).

S**socket**

Type of file used for network communication between processes.

software unit

Discrete set of software statements that implements a function; usually a sub-component of a CSC.

Solaris

Name of the operating system used on Sun Microsystems hardware.

SQL

Structured Query Language; a language for manipulating data in a relational database.

T**tar**

Tape archive. UNIX command for storing or retrieving files and directories. Also used to describe the file or tape that contains the archived information.

TCP/IP

Transmission Control Protocol/Internet Protocol.

thread

Short for a processing thread, which is an execution path. A process may be single or multi-threaded. In a multi-threaded process all threads share a single address space but have independent execution paths.

U**UNIX**

Trade name of the operating system used by the Sun workstations.

URL

Uniform Resource Locator.

W**WAN**

Wide Area Network.

wfdisc

Waveform description record or table.

Index

Symbols

/etc/inetd.conf
configuration [107](#), [108](#)
/etc/services
configuration [107](#), [108](#)

A

affiliation [124](#), [126](#), [136](#), [146](#)
alphasite [118](#), [124](#), [136](#), [146](#)
architectural data flow [11](#)
auth_<inst>.par [123](#)
authentication
 adding data providers [154](#)
 adding forwarding destinations [162](#)
 configuration [134](#)
 performance [6](#)
 updating [144](#)

C

CD-1.0 [2](#)
CD-1.1
 protocol [2](#)
CDS
 install [137](#)
 restart [140](#)
 shutdown [137](#)

cds_idc_env [10](#), [113](#), [115](#)
CDS CD-1.1
 overview [2](#)
cds.par [116](#)
configuration
 authentication [134](#)
 Connection Manager (ConnMgr) [117](#)
 Connection Manager Server
 (ConnMgr_server) [117](#)
 Connection Originator (ConnOrig) [119](#)
 Data Center Manager (ForeMan) [133](#)
 Data Parser (DLParse) [123](#)
 data stores [111](#)
 Exchange Controller (ExCltr) [120](#)
 file system [108](#)
 firewall [109](#)
 Frame Exchange (FrameEx) [120](#)
 Frame Store [129](#)
 internet daemon [107](#)
 log files [111](#)
 subsystem level [116](#)
configuration issues [7](#)
Connection Manager (ConnMgr) [8](#), [16](#)
 changing [143](#)
 configuration [117](#)
 error messages [43](#)
 manual start-up [22](#)
Connection Manager Server
 (ConnMgr_server) [8](#), [16](#)
 configuration [117](#)
 error messages [63](#)
 manual start-up [21](#)

▼ Index

Connection Originator (*ConnOrig*) [9](#)
 adding data providers [153](#)
 adding forwarding destinations [157](#)
 configuration [119](#)
 log file [39](#)
 manual start-up [22](#)
 monitoring [37](#)
 connection state transitions [19](#)
 connmgr_idc.par [117](#)
 ConnOrig_<inst>.par [157](#)
 ConnOrig.par [119](#)
 connsvr_idc.par [117](#)
 ctrlr_fex_<instance>.par [121](#)
 ctrlr_fex_IDC_<dest>.par [158](#)
 ctrlr_fex_idc_<dest>.par [144](#)
 ctrlr_fex_idc_sta.par [143](#)
 customization [7](#)

D

database [135](#)
 adding data providers [154](#)
 updating [145](#)
 Data Center Manager (*ForeMan*) [9](#)
 adding forwarding destinations [160](#)
 command line [20](#)
 configuration [133](#)
 error messages [82](#)
 log file [42](#)
 manual start-up [24](#)
 monitoring processes controlled by [37](#)
 data flow symbols [v](#)
 Data Parser (*DLParse*) [9](#)
 adding data providers [154](#)
 configuration [123](#)
 error messages [77](#)
 log file [41](#)
 manual start-up [24](#)
 monitoring [37](#)
 data providers
 adding [140](#)
 short course for adding [152](#)

data stores
 configuration [111](#)
 DBMS [12](#)
 development status [5](#)
 disk space requirement [12](#)
 dl<inst>.par [123](#)
 dlfile [124](#)
 dlman [118](#), [136](#), [147](#)

E

environment
 adding data providers [154](#)
 adding forwarding destinations [163](#)
 altering [151](#)
 creating [112](#), [115](#)
 script for creating [A2](#)
 error messages [43](#)
 error recovery [103](#)
 Exchange Controller (*ExCltr*) [9](#), [18](#)
 adding data providers [153](#)
 adding forwarding destinations [158](#)
 configuration [120](#)
 consumer setup [143](#)
 error messages [94](#)
 forwarding setup [144](#)
 manual start-up [23](#)
 Exchange Controller (*ExCtrl*)
 log file [40](#)
 monitoring [37](#)
 executable files
 location [110](#)

F

file system
 configuration [108](#)
 firewall [12](#)
 configuration [109](#)
 ForeMan_<inst>.par [133](#)

forwarding [6](#)
 adding destinations [155](#)
 Frame Exchange (*FrameEx*) [10](#), [18](#)
 adding data providers [153](#)
 adding forwarding destinations [158](#)
 configuration [120](#)
 consumer setup [143](#)
 error messages [89](#)
 forwarding setup [144](#)
 manual start-up [23](#)
 monitoring [37](#)
 Frame Store
 adding data providers [153](#)
 adding forwarding destinations [156](#)
 changing [141](#)
 configuration [129](#)
fset_maker [A7](#)

H

hardware mapping [106](#)

I

inbound data
 monitoring [32](#)
index.txt entries [134](#)
inetd [16](#)
 installation
 short course [136](#)
 installation procedures [106](#)
instrument [124](#), [136](#)
 internet daemon configuration [107](#)

L

log files
 attributes [29](#)
 configuration [111](#)
 monitoring with [32](#)

log level [33](#)
 3 (error conditions) [36](#)
 5 (steady-state operations) [34](#)
 6 (frame transactions) [36](#)

M

maintenance [29](#)
Mkloop [A8](#)
Mkloop.par [A8](#)
 monitoring [32](#)
 processes controlled by Data Center Manager [37](#)

P

par file
 hierarchy [113](#)
 naming convention [114](#)
 performance characteristics [5](#)
port_idc.par [118](#)
 problem reporting [104](#)
 Protocol Check (*ProtoCheck*) [10](#)
 error messages [102](#)

R

run_idc_connmgr [10](#)
run_idc_connsrv [10](#)
run_idc_dcmgr [10](#)

S

security [30](#)
sensor [124](#), [126](#), [136](#), [147](#)
setup_env [112](#)
setup_idc_env [112](#)

▼ Index

shutdown
 automated [25](#)
 manual [27](#)
site [125](#), [136](#), [148](#)
sitechan [124](#), [127](#), [136](#), [148](#)
software environment [12](#)
software release
 obtaining [106](#)
start-up
 automated [16](#)
 manual [21](#)
state transitions [18](#)
subsystem
 adding forwarding destinations [155](#)

T

typographical conventions [vi](#)

U

user account [106](#)

W

WAN connections [12](#)
wfconv [124](#), [127](#), [136](#), [149](#)
wfdisc [124](#)
wfproto [124](#), [128](#), [136](#), [150](#)